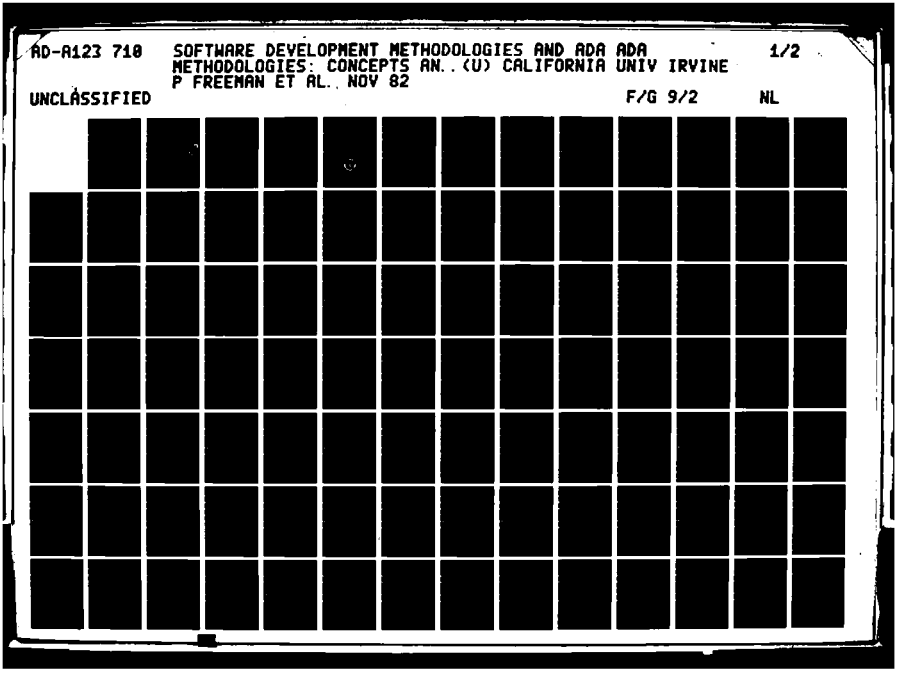
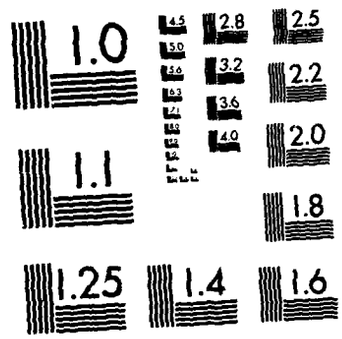

AD-A123 710
SOFTWARE DEVELOPMENT METHODOLOGIES AND ADA ADA METHODOLOGIES: CONCEPTS AN..(U) CALIFORNIA UNIV IRVINE P FREEMAN ET AL. NOV 82
1/2
UNCLASSIFIED
F/G 9/2
NL



Software Development Methodologies and Ada

Ada Methodologies:

Concepts and Requirements

Ada Methodology Questionnaire Summary

Comparing Software Design Methods for Ada: A Study Plan

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER	2 GOVT ACCESSION NO. 40-A123710	3 RECIPIENT'S CATALOG NUMBER
4 TITLE (and Subtitle) Software Development Methodologies and Ada; Ada Methodologies Concepts and Requirements, Ada Methodology Questionnaire Summary, Comparing Software Design	5 TYPE OF REPORT & PERIOD COVERED Draft Report 6 PERFORMING ORG. REPORT NUMBER
7 AUTHOR(s) Peter Freeman and Anthony I. Wasserman	8 CONTRACT OR GRANT NUMBER(s)
9 PERFORMING ORGANIZATION NAME AND ADDRESS University at California, Irvine	10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11 CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office (AJPO) Arlington, Virginia	12 REPORT DATE November, 1981 13 NUMBER OF PAGES
14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15 SECURITY CLASS. (of this report) U 15a DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18 SUPPLEMENTARY NOTES

This document is also known as "Methodman."

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Software
Ada
Programming Support Systems
Software development
Management information systems.

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

The document is divided into three sections as indicated by its subtitles. The concepts of software development methodologies as they relate to the computer programming language Ada, are presented in this first draft document.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada Methodologies:

Concepts and Requirements

Department of Defense

Ada Joint Program Office

Ada[TM] Methodologies: Concepts and Requirements

November 1982

[TM]Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office).

PREFACE

As I observed in Ada® Letters ("The Need for a Programming Discipline to Support the APSE", Vol. I, No. 4, pp 21-23, May/June 82), we will not realize the full potential of Ada until we are able to define a software development methodology complete with management practices which can in turn be supported by automated tools. I announced in that paper that Professors Freeman and Wasserman had agreed to prepare a first draft of a "Methodman". This document is the promised draft.

Although I believe that Professors Freeman and Wasserman have done a superb job of defining the desirable characteristics of a methodology to support the software development process, we recognize that it is incomplete and must be refined. Just as Strawman was refined to produce Steelman, and Pebbleman was refined into Stoneman, this document is being circulated for comment that will assist in the refinement process.

Many software activities (both technical and managerial) are independent of the programming language. Indeed, although the Ada language and the APSE provide a basis for better support for a software discipline, the development of a total life-cycle methodology expands the scope of the Ada Program. On October 8, 1982, Dr. Edith W. Martin, Deputy Under Secretary of Defense for Research and Advanced Technology, announced plans for a DoD Software Initiative. This initial version of "Methodman" may be viewed as an initial function of the initiative. It demonstrates the essential role that the Ada Program will play in the initiative and concurrently, the need for the initiative to press for technology innovation beyond Ada.

The Ada Program has received substantial benefit from public interaction. The distribution of this draft is another important step in continuing that interaction. It is also indicative of the professionalism of Peter Freeman and Tony Wasserman that they consent (as did Dave Fisher and John Buxton) to subject their work for public scrutiny and comment.

Constructive comments should be sent to the Ada Joint Program Office, Suite 1210, 801 N. Randolph St. Arlington, VA 22203.

Larry E. Druffel, Lt. Colonel, USAF
Director
Ada Joint Program Office

UNIVERSITY OF CALIFORNIA

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO

SANTA BARBARA • SANTA CRUZ

November, 1982

This document has benefited greatly from the many valuable comments that we have received on its previous drafts. We are thankful to everyone who gave us both written and verbal comments during the preliminary review process, and appreciate their contributions. These people include: B.N. Barnett, L.A. Belady, M. Brodie, R. Bruno, G. Estrin, R. Fairley, H. Fischer, R.L. Glass, S. Gutz, H. Hart, H. Hess, R. Houghton, C.A. Irvine, M.A. Jackson, L. Johnson, E. Koskela, J. Lancaster, J. Larcher, B. Liskov, P.C. Lockemann, I. Macdonald, J.B. Munson, E.J. Neuhold, D.L. Parnas, L. Peters, K.T. Rawlinson, D.J. Reifer, W. Riddle, C. Rolland, T. Standish, H.G. Stuebing, W. Tichy, R. Van Tilburg, L. Tripp, C. Tully, J. Wileden, M. Zelkowitz, and N. Zvegintzov.

We included a selected bibliography in an earlier draft, but have decided against its inclusion in this report, since a bibliography would have recognized the work of some while omitting that of others. It should be apparent that our ideas have been shaped by a vast amount of work both inside and outside DoD over the past 15 years, and that our role here was to consolidate this body of work and to establish a framework from which methodologies could be developed and enhanced. We gratefully acknowledge this debt to our colleagues who have contributed to the present understanding of software development methodologies.

We especially wish to thank Lt. Col. Larry Druffel, who sponsored this effort, and provided the framework in which we could accomplish the work. Finally, we appreciate the assistance of USC Information Sciences Insitute for handling the administrative aspects of this work.

Anthony Ira Wasserman

Anthony I. Wasserman
Medical Information Science
University of California, San Francisco

Peter Freeman

Peter Freeman
Information and Computer Science
University of California, Irvine

TABLE OF CONTENTS

1. INTRODUCTION 1

2. RATIONALE 1
- 2.1 Analysis and modeling 4
- 2.2 Functional specification 4
- 2.3 Design 5
- 2.4 Implementation 5
- 2.5 Validation and Verification 5
- 2.6 Management Procedures 6

3. REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY 6
- 3.1 Support for Entire Development Process and Transitions between Phases 7
- 3.2 Support for Communication among Interested Persons 8
- 3.3 Support for Problem Analysis and Understanding 8
- 3.4 Support for Top-Down and Bottom-Up Development 8
- 3.5 Support for Validation and Verification 8
- 3.6 Support for Constraints 9
- 3.7 Support for the Software Development Organization 9
- 3.8 Support for System Evolution 9
- 3.9 Automated Support 9
- 3.10 Support for Software Configuration Management 10
- 3.11 Teachability and Transferability 10
- 3.12 Open Ended 10

4. RELATIONSHIP TO ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) 10

5. METHODOLOGY EVALUATION 11
- 5.1 Technical Characteristics 11
- 5.2 Usage Characteristics 13
- 5.3 Management Characteristics 14
- 5.4 Economic Characteristics 15

6. CONCLUSION 15

1. INTRODUCTION

This document rationalizes the need for the use of coherent software development methodologies in conjunction with Ada and its programming support environments (APSE's) and describes the characteristics that such methodologies should possess. It is recognized that software development, particularly for embedded systems, is increasingly done in the context of overall systems development, including hardware and environmental factors. While there is a strong need for integrated systems engineering, this document focuses on the software issues only.

Emphasis is thus given to the *process* by which software is developed for Ada applications, not just with the language or its automated support environment. The development activity yields a collection of work products (including source and object versions of Ada programs). These work products are valuable not only through the development phase, but also through the entire lifetime of the system as modifications and enhancements are made to the system.

The analysis, design, and development of complex systems, such as those to be programmed in Ada, must be controlled through a collection of management procedures and technical methods. Differences in organizational structures, applications, and existing approaches, however, make it impractical to prescribe a single methodology that can be uniformly followed.

Accordingly, this paper identifies requirements for software development methodologies, as was done in the sequences of documents leading to the "Steelman" and "Stoneman" reports. A preliminary version of these requirements has been used to evaluate some existing methodologies for software development. These requirements can serve as a basis for evolving methodologies and for creating new ones oriented to the special problems of embedded systems to be implemented in Ada. The emphasis, then, is on the conceptual basis for software development methodologies.

As with the work leading to the design of Ada and the specifications for Ada Programming Support Environments, this work builds upon, but is not constrained by, the current set of methodologies and development practices. Instead, this document sets down some underlying principles. Thus, there are no references to specific techniques or to existing standards, based on the belief that improved methodologies can be developed from the foundation established here. Of course, such standards and techniques have influenced the requirements identified here.

An important assumption throughout this document is that the methodology used for creating systems should be preeminent over the tools used in it. In other words, tools should support a methodology, and not the other way around.

There is also very little emphasis on Ada itself, except in the discussion of implementation. Other aspects of the development methodology are described on the assumption that Ada is the target programming language. However, most of the concepts are based on currently understood notions of software engineering and programming methodology, and are therefore not tightly coupled to the programming language. Indeed, many of the requirements for a software development methodology are largely independent of the target programming language.

2. RATIONALE

A key assumption of modern software development practices is that increased effort in the earlier stages of development will be reflected in reduced costs for testing and evolution* (maintenance). Such effort is intended both to prevent errors from being introduced into the system, and to detect any such errors at the earliest possible time. The resulting software will

*The word "evolution" is used rather than "maintenance" throughout this document to refer to the three activities of repair, adaptation, and enhancement that may occur after initial development. The term "evolution" is intended to better describe the actual situation without the traditionally negative connotation of "maintenance."

be of higher quality and will be more likely to fulfill the needs of its users.

The unifying notion is that of a *coherent methodology*, a system of technical methods and management procedures that covers the entire development activity. A methodology can be supported by automated tools; the collection of available tools provides a "programming support environment," which can, among other things, aid developer communication and productivity. Within a methodology, it is important to be able to review the progress of the work at various intermediate checkpoints. In this manner, it becomes possible to identify problems in projects at earlier stages and to take corrective action.

An important concept in discussing methodologies is the "life cycle", a model of the activities that comprise the software development and evolution of a system. There are numerous life cycle models in use, with still others described in the literature. Each of these models forms a framework for describing the steps of system development and the products that are produced at each step. Rather than adhere to a *single* existing model, this document follows a model that is *representative* of many (but not all) of the models, so that one can easily relate it to others.

The specific phases and names vary from one model to another, but typically include analysis, functional specification, design, implementation, validation, and evolution, defined as follows:

Analysis - a step concerned with understanding the problem and describing the activities, data, information flow, relationships and constraints of the problem; the typical result is a requirements definition;

Functional specification - the process of going from the statement of the requirements to a description of the functions to be performed by the system to process the required data; functional specification involves the *external* design of the software;

Design - the process of devising the *internal* structure of the software to provide the functions specified in the previous stage, resulting in a description of the system structure, the architecture of the system components, the algorithms to be used, and the logical data structures;

Implementation - the production of executable code that realizes the design specified in the previous stage;

Validation - the process of assuring that *each* phase of the development process is of acceptable quality and is an accurate transformation from the previous phase+;

Evolution - the ongoing modifications (repair, adaptation to new conditions, enhancement with new functions) to a system caused by new requirements and/or the discovery of errors in the current version of a system.

This sequence of phases separates analysis and functional specification activities, based on the observation that the product of analysis is a functional specification, but that the functional specification may only address a portion of the problem that has been analyzed. (Note that "specification" is used both as a verb to denote a process and as a noun to denote a work product.)

Throughout development and evolution, there are aspects of management and communication, including documentation, validation, budgeting, personnel deployment, project review, scheduling, and configuration management, that serve to tie the stages together and provide the organizational environment in which the technical procedures can be made effective.

+Note that validation is not simply a single phase, but rather a step performed as part of each phase. Thus, validation of design is the process of determining that the design is a valid transformation from the functional specification.

The combination of technical procedures with management techniques should create a synergistic effect in which the resulting process provides significantly greater improvement in the production of software than would be provided by either the technical or the management elements alone.

In short, one cannot choose a tool, a management practice, or any other element of the total environment without considering that element in its relation to the other parts of the development system. This concept is illustrated by Figure 1.

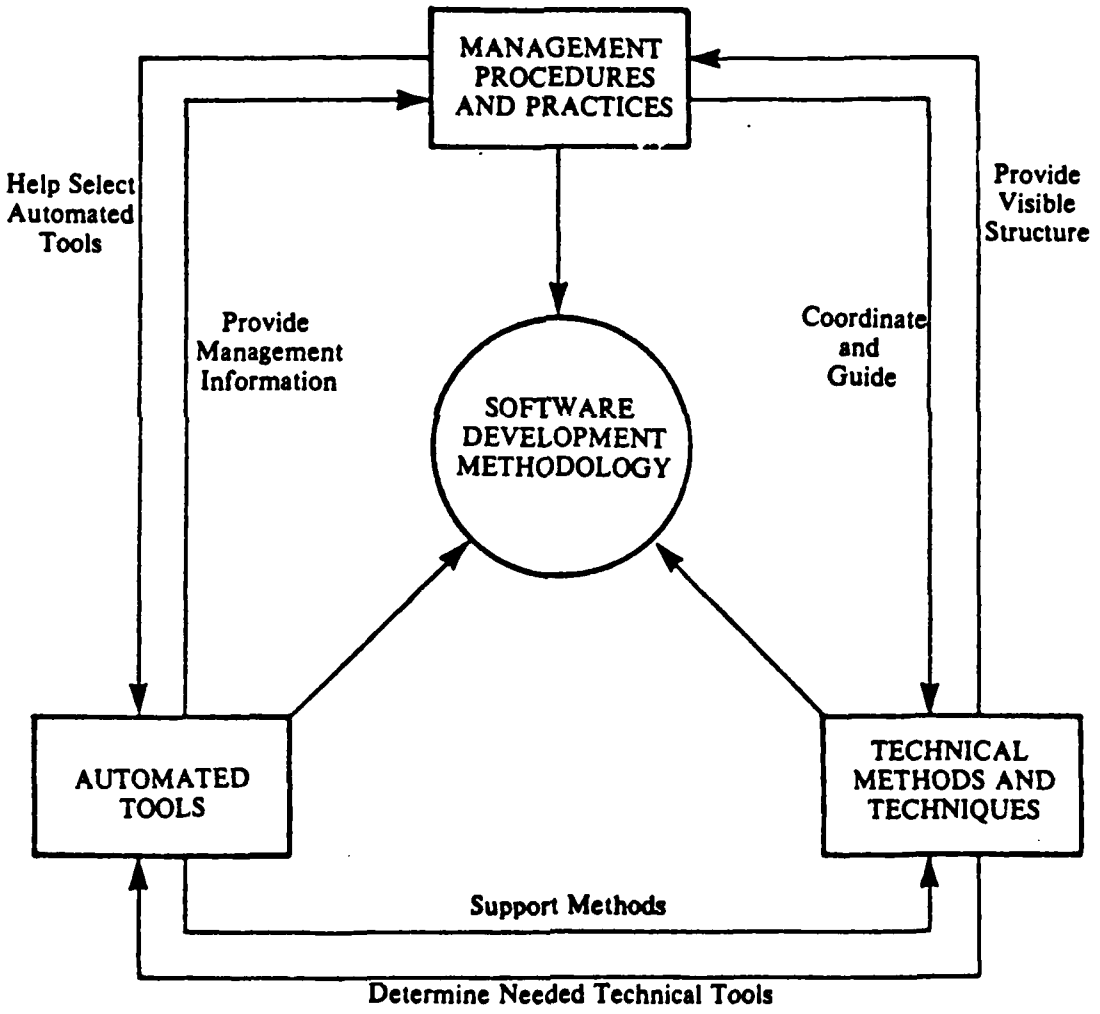


Figure 1 SOFTWARE DEVELOPMENT METHODOLOGY

It is essential that management and technical elements be synergistic. It can be seen from Figure 1 that management methods provide guidance in the use of technical methods, while the technical methods serve to provide the intermediate results that make management possible. The importance of this concept in the context of a methodology is that one cannot look at technical or management elements alone, but rather must look at their combination. Before proceeding with the requirements for a methodology to support Ada, it is useful to describe the roles of various development phases in somewhat greater detail.

2.1 Analysis

Analysis of the problem at hand is the essential first step of any software development activity. Without such analysis, it is impossible to proceed; furthermore, an inadequate job of analysis is virtually certain to lead to project failure, since poor understanding of the problem makes it impossible to produce a good specification.

Successful analysis involves communication with users and customers for the system, who can describe their needs. Analysis also involves communication with the eventual developers of the system, who must be able to evaluate implementation feasibility and to describe any design or implementation constraints.

Because of the complexity of systems, key tools for analysis must support problem decomposition, through any of a variety of schemes, including: procedural decomposition, data abstraction, data flow, processing sequence, or transaction modeling. Graphical notations are especially helpful in showing the interrelationships of system components to one another and in facilitating the communication process.

A particularly effective method for analysis is *modeling*, representing the problem and/or the real world situation in a formal (mathematical or graphical) notation. The evolving model can be used as the basis for communication and for understanding the tasks that will comprise the system. In some instances, an *executable* model, or a prototype of some part of the system, may be built to assist in the analysis process. In others, properties of the model may be exploited to learn more about the subject (for example, whether certain activities may occur in parallel).

2.2 Functional Specification

A functional specification is a description of "what" the system will do. Whereas analysis serves to describe the problem and to identify requirements, the functional specification is the first statement of the system's intended behavior. Thus, it contains a statement of the system functions, the external interfaces, the required data items, and requirements upon performance, design, and implementation.

Functional specifications have many different roles within the software life cycle, including the following:

- The functional specification is a means for precisely stating the software system requirements. At this stage, the technical realization of the system model is documented in as much detail as possible. The functional specification can be compared against the requirements definition to ascertain the correspondence between the specification and the needs.
- The functional specification provides insight into the problem structure and is used during the design phase as a checkpoint against which to validate the design. Typically, there will be an iteration between specification and design, as insight into some of the system construction problems help to clarify the functional specification.
- The functional specification is the basis against which validation is performed. Clearly, one cannot validate a program in the absence of an unambiguous functional specification. Such a functional specification is essential whether validation is carried out through acceptance testing or through formal proof of program correctness.
- Modifications and enhancements to a system throughout its operational lifetime require an understanding of the system functions, as documented in the functional specification.

During evolution, the functional specification can help to locate those system functions that must be changed, and can then be revised accordingly.

The implication of this multifaceted role is that a functional specification must be able to serve (to some degree) each of these four different functions. In practice, this means that functional specifications must have both a formal and an informal component.

Throughout the remainder of this paper, the term "specification" refers to a functional specification. This usage should be distinguished from a more general usage in which each stage is a "specification" of what is to be done in the subsequent stage. In that usage, a detailed description of the logic of a sorting algorithm would be considered as a specification for Ada code to be written. Thus, this document refers to "the design" where others might use the term "the design specification."

2.3 Design

The process of software design permits the developer to determine more precisely the feasibility of implementing the functional specification, to analyze alternative system structures, to develop algorithms for the solution of the problem, and to define detailed constraints upon the implementation. In summary, it is a stage at which the primary concern is with the internal structures from which the software will be built.

The internal design activity can be separated into two phases: architectural design and detailed design. (In very large systems, one can separate design into more than two phases.) Architectural design is concerned with recognition of the overall software structure and the interconnection of program pieces, while detailed design is more concerned with the selection of algorithms and data structures that are appropriate to the fulfillment of specific system functions.

One of the key goals of the design process is to simplify the subsequent stages of coding and testing. At the end of the design phase, virtually all of the key decisions concerning program organization, logical data structures, and processing algorithms will have been made, with the intent of making code production into a straightforward transformation of the design into an executable representation.

The output of the design activity is a *software blueprint* that can be used by the Ada programmer(s) to implement the system without having to refer back to the functional specification and without having to make unwarranted assumptions about the requirements.

2.4 Implementation

Implementation of systems, in this instance, involves the production of executable Ada code. The code should reflect the structure of the design and perform the function(s) specified for the system. The code should adhere to the precepts of structured programming, with emphasis on comprehensibility of code. It is important to note that coding is only a small portion of the overall software development activity; good coding cannot make up for poor analysis or design.

In general, the comprehensibility of programs written in a high-level language can be achieved through uniformity of programming style, use of mnemonic names for procedures and data, judicious inclusion of comments, and avoidance of unrestricted control flow. Comprehensibility of Ada programs is further enhanced by use of the information hiding properties of modules (*package*), minimization of interaction between tasks and exceptions, minimization of the different choices for *digits* and for *delta* in a single program, and careful use and documentation of tasking statements such as guarded *select, delay* and *terminate.*

2.5 Validation and Verification

Validation is the process of determining that a system correctly performs those functions described in the functional specification. Verification is a process to ensure that each phase of the development process correctly carries out the intent of the previous step. One must verify the code against the system design, which in turn has been verified against the functional specification for the system. Together, they provide assurance of system quality.

Validation of programs (code) may be done through either testing or formal proof of correctness. Although there has been much work on mathematical proofs of program correctness, most code is verified and validated through testing, which may be described as a series of controlled experiments to provide empirical evidence that a program behaves properly (and provides the desired results for broad classes of anticipated inputs).

Testing is normally done in three stages: module testing, integration testing, and acceptance testing. In module testing, individual program units are tested for correctness. In integration testing, two or more modules are joined and tested together, to see if they work properly together and to make certain that the interfaces mesh. Finally, acceptance testing determines whether the system conforms to its functional specification.

In general, errors found in module and integration testing reflect errors made during design or implementation, while errors found during acceptance testing reflect specification errors -- incomplete, inconsistent, incorrect, or ambiguous statements of what the system was to do. The most serious aspect of this situation is that the errors that were made first are detected last! An error in the requirements definition may not be caught until the entire system has been constructed and tested; such an error may require massive changes in the system design and implementation. It is for this reason that analysis and design errors are the most expensive kind of errors, and that efforts such as formal reviews of design and code have a significant payoff in terms of development costs.

2.6 Management Procedures

As noted earlier, a software development methodology is actually a blend between a collection of technical procedures and a set of management techniques that can result in effective deployment of project personnel, predictability of project schedule, budget, and outcome, accurate estimation of software properties, and the final result of a high-quality system that meets the needs of its users throughout the lifetime of the system.

Management of software development involves both management of people and management of the software product. The former involves issues of supervision of individual and project progress, and selection of appropriate team organizations and individual assignments. The latter focuses on deciding when a system is ready to be released and controlling the means by which it (and its subsequent versions) are released and modified. Furthermore, the management activity includes selection and revision of the technical procedures that are to be used by the systems organization.

The discipline provided by software development techniques leads to an environment in which management becomes possible, primarily because these techniques require the creation of intermediate products, e.g., specifications and designs, that can be reviewed and used to measure progress. Discipline refers to the adherence to a systematic procedure for the process of software production, a procedure that can be followed and repeated for a large class of software projects.

3. REQUIREMENTS FOR A SOFTWARE DEVELOPMENT METHODOLOGY

The goal of a software development activity is the effective creation of a set of work products, comprising an operational system and its supporting documents. For every software system, there are desirable qualities, such as reliability, correctness, evolvability, and efficiency, just to name several of the most common. A software development methodology should assure, to the greatest extent possible, that these system qualities are achieved. Furthermore, the methodology should make it possible to decide among different system qualities, rather than restricting the choice.

Many approaches have been proposed and used for creating systems. There is every indication that these approaches will be refined and that new approaches will be introduced in the future. Furthermore, it seems clear that many of these approaches can be successfully used

to develop systems implemented in Ada. Our notion of a methodology is that of a *system* of methods and tools, carefully chosen and integrated to support both the development and evolution processes.

The purpose of this section, then, is to establish a framework for development methodologies through a set of requirements for a methodology. The list of requirements is given first, and each requirement is then discussed at greater length.

A methodology should:

1. cover the *entire* development process, simplifying transitions between project phases;

2. enhance communication among all interested persons at all stages of development;

3. support problem analysis and understanding;

4. support both top-down and bottom-up approaches to software development;

5. support software validation and verification through the development process;

6. facilitate the capture of design, implementation, and performance constraints in the system requirements;

7. support the software development organization;

8. support the evolution of a system throughout its existence;

9. be supported by automated aids;

10. make the evolving software product visible and controllable at all stages of development;

11. be teachable and transferable;

12. be open-ended.

3.1. ENTIRE DEVELOPMENT PROCESS

A methodology should cover the *entire* development process. It does little good to have a methodology for software design if there is no systematic procedure to produce the functional specification used for the design and the Ada program(s) that must be created from the design. In other words, it should address all of the phases discussed in Section 2, from analysis through evolution.

The methodology should facilitate transitions between phases of the development cycle. When a developer is working on a particular phase of a project (other than requirements analysis), it is important to be able to refer to the previous phase and to trace one's work. At the design stage, for example, one must make certain that the architecture of the software system provides for all of the specified functions; one should be able to identify the software module(s) that fulfill each system function. During implementation, it should be easy to establish a correspondence between modules in the system design and program units, and between the logical data objects from the design stage and the physical data objects in the program.

It is important to note that one must be able to proceed not only forward to the next phase of the life cycle, but also backward to a previous phase so that work can be checked and any necessary corrections can be made. This phased approach to software development makes it clear that information lost at a particular phase is generally lost forever, with an impact on the

resulting system. For example, if an analyst fails to document a requirement, it will not appear in the functional specification. Eventually, during acceptance testing (or perhaps during system operation), that failure will be recognized and it will be necessary to make modifications to the system.

3.2 ENHANCE COMMUNICATION AMONG INTERESTED PARTIES

A methodology should enhance communication among all of the different persons involved in a software development project at all of the different stages of development. Among the paths of communication that should be supported are those among developers and users, developers and customers, developers and their managers, and among developers themselves.

Written documents serve to support this communication by providing a record of decisions and agreements. At times, such documents may be informal, simply recording a piece of information or an agreement. This approach is particularly useful when a document is intended for someone without specialized knowledge of software development. However, precise communication among technically knowledgeable individuals is enhanced by the use of formal notations, including graphical representations and specialized languages. Examples of such notations are data flow diagrams, HIPO diagrams, algebraic specification of data types, and Ada. The methodology should prescribe the forms of documentation and representation that will be used for this technical communication at all stages of the development process.

3.3 SUPPORT FOR PROBLEM ANALYSIS AND UNDERSTANDING

A methodology should support effective problem-solving techniques. It should encompass intellectual processes to support problem decomposition.

Modeling is a particularly important aspect of this objective. There exist techniques for activity and data modeling. Formal modeling techniques should be based upon a suitable set of primitives for the application domain. For embedded systems, concepts such as external interfaces and concurrent processes should be included. The resulting model should serve to answer questions about the problem domain and can be used to define the scope of interest for the system. Unless one has suitable tools for analysis of the problem, one cannot easily produce a functional specification or a satisfactory system.

Problem solving techniques are used to decompose the problem and to create the model. Techniques such as data abstraction, data flow diagrams, functional decomposition, transaction modeling, and state machines have been effectively used for this purpose in the past. The goal of this phase is to determine the structure of the problem -- the interrelation of the parts of the problem.

3.4 SUPPORT FOR TOP-DOWN AND BOTTOM-UP DEVELOPMENT

The methodology should support a variety of different approaches to system design and development. While systems are often modeled or analyzed using methods of top-down decomposition, there are typically low-level constraints, such as interface requirements, for which a bottom-up approach is necessary. Efforts to reuse software designs and/or code, including the use of packages, also leads to the use of bottom-up techniques to develop software. Finally, design constraints, such as performance requirements, necessitate the creation and testing of low-level aspects of a system before many of the high-level aspects have been developed.

Thus, the methodology must not constrain the development organization to follow either a pure bottom-up or a pure top-down approach to software development, but should allow any combination of the two in system design and implementation.

3.5 SUPPORT FOR VALIDATION AND VERIFICATION

The methodology must support determination of system correctness throughout the life cycle. System correctness encompasses many issues, including not only the correspondence between the results of one stage of development and the previous stage, but also the extent to which the system meets user needs. Accordingly, the methodology must not only be concerned

with techniques for validation of the complete system, but also must give attention to obtaining the most complete and consistent verification of each work product throughout the development. For example, the methods used for analysis and specification of the system should make it possible to trace later system development back to the requirements and functional specification.

The methodology must prescribe a strategy for assurance of system quality. Test planning, document review, design and code walkthroughs and/or inspections should be integrated in the methodology, with emphasis given to early error detection and correction. A test plan should establish standards for test coverage, a means for measuring that coverage, and the acceptable quality criterion.

3.6 SUPPORT FOR CONSTRAINTS

A methodology should facilitate the inclusion of design, implementation, and performance constraints in the system requirements. Embedded systems to be constructed in Ada often have severe requirements on memory utilization, real time response, use of specific machine-dependent features, or integration with other hardware or software systems. Experience has shown that severe constraints have a major detrimental effect on system development.

It should be possible to state these requirements using the methodology. Furthermore, it should allow them to be incorporated into each subsequent stage and verified. This requirement may necessitate the use of analytical tools and/or construction of prototype systems.

3.7 SUPPORT FOR THE SOFTWARE DEVELOPMENT ORGANIZATION

The methodology must, above all, support the intellectual efforts of the designers and other technical people. Beyond this, it should support the software development organization that has been chosen for the project at hand. It must be possible to manage the developers and the developers must be able to work together. This requirement implies the need for effective communication among analysts, developers, and managers, with well-defined steps for making progress visible throughout the development activity. The intermediate products generated by the methods and tools, such as a detailed design or an acceptance test plan, can be reviewed by the organization so that progress can be effectively measured and so that quality can be assured.

The methodology must support the management of the project. It should include methods for cost estimation, project planning (scheduling), and staffing. It should also specify methods for ongoing review of project progress, such as design walkthroughs and code inspections. The management procedures should maintain a project handbook and library, showing project history, project plans, and the evolving software product(s). Finally, the management procedures should identify an evolving set of standards and conventions that can be applied to all projects.

3.8 SUPPORT FOR SYSTEM EVOLUTION

The methodology should support the eventual evolution of the system. Systems typically go through many versions during their lifetimes, which may last eight to ten years or more. New requirements arise from changes in technology, usage patterns, or user needs, and these changes or additional requirements must be reflected in a modified system. In many ways, the evolution activity is a microcosm of the development process itself. The development methodology can assist this evolutionary activity by providing accurate external and internal system documentation, and a well structured software system that is easily prototype and modified by those making the system changes.

3.9 AUTOMATED SUPPORT

Wherever possible, the methodology should be supported by automated tools that improve the productivity of both the individual developer and the development team. This collection of tools, and the way in which they are used, constitute a "programming support environment." (Note that the word programming is not used in the sense of "coding" here, but rather in a more general sense.) A partial set of recommended tools for such an environment

are described in the "STONEMAN".

The tools should be integrated so that they may communicate via a common database and so that they can work effectively with one another. Furthermore, these tools should be linked to the methods so that the tools support the management methods and technical practices.

3.10 SOFTWARE CONFIGURATION

The methodology should maintain the visibility of the emerging and evolving software product and its supporting work products. All of these items should be placed in the database of the associated Ada Programming Support Environment.

The underlying notion is that of software configuration management. All of the components of a software development project must be identified, collected, and controlled in order to assure proper distribution of the finished system and its supporting work products, as well as to assist in evolution of the product and version control.

The methodology must be applicable to a large class of software projects. While it is clear that different methodologies will be needed for different classes of systems and for different organizational structures, an organization should be able to adopt a methodology that will be useful for a sizeable number of programs that they will build. Certainly, it makes little sense to develop a methodology for each new system to be built.

3.11 TEACHABILITY AND TRANSFERABILITY

The methodology must be teachable. Even within a single organization, there will be a sizeable number of people who must use the methodology. These people include not only those who are there when the methodology is first adopted, but also those who join the organization at a later time. Each of these people must understand specific techniques that comprise the technical aspects of the methodology, the organizational and managerial procedures that make it effective, automated tools that support the methodology,and the underlying motivations for the methodology.

Transferability of the methodology is greatly aided by teaching materials, including user documentation, organized courses, exercises, and examples.

3.12 OPEN-ENDED

A methodology should be open-ended. It should be possible to introduce new technical and managerial methods, as well as to create new tools, and thereby modify the methodology. Techniques and tools evolve through a process of "natural selection", whereby newer, more effective techniques and tools, supplement or replace existing ones.

4. RELATIONSHIP TO ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE)

The "STONEMAN" Report defines three levels of programming support environment for Ada.

KAPSE - A Kernel APSE supporting basic functions of operating systems, database, and communication support. This Kernel can provide tool portability from one computer system to another.

MAPSE - A Minimal APSE providing a methodology-independent set of essential tools to support the Ada programmer. These tools include language editors, translators, configuration management, loading, linking, and static and dynamic analysis tools.

APSE - fuller support for development of Ada programs, including tools for requirements, specification, design and management. An APSE offers a coordinated and complete set of tools which is applicable at all stages of the system life cycle.

It is the full APSE that provides the linkage to specific methodologies for system development. The APSE exists to support the methodology (not the other way around), since the toolset should be a means to an end and not an end in itself.

Specifically, STONEMAN describes a database as the central feature of an APSE system. (It should be noted that this usage of "database" differs from its more traditional data-processing usage.) The database acts as the repository for all information associated with each project throughout the project life cycle.

The automated tools of the methodology (see 3.8) comprise the APSE. They are used to create, modify, analyze, transform, and display objects in the database. The workproducts produced by the methodology are included in the database for a project.

Thus, the APSE and the methodology have a symmetric relationship. The methodology defines the tools at the APSE level (building upon the MAPSE), while the APSE provides the central repository needed to support teamwork and quality control in the methodology.

The methodology and the tools, in turn, are part of a more general software development environment, as shown in Figure 2. This broader notion of environment includes the software developers, the organization to which they belong, and the physical workspace in which software development takes place.

There are many influences within the workplace that affect the productivity of software developers, including access to computers, privacy and noise levels in working areas, ergonomic considerations of terminals, and availability of reference materials, including books and journals. There is little doubt that such factors can have significant bearing on the productivity of developers, regardless of methodology and tools, and further study must be done on identification and measurement of the most significant factors so that every possible step is taken to enhance the ability of software development organizations to produce the best possible systems in the minimum interval of time.

5. METHODOLOGY EVALUATION

In a strict sense, *every* software development organization already has a methodology for building software systems. While some software is developed according to modern practices of software development, most of it is built in an *ad hoc* way. Accordingly, it is best to view the discussion of software development methodologies from the perspective of *changing* current practices, replacing them with new techniques that improve the process of software development and the quality of the resulting products.

The process of change requires an understanding of the strengths and weaknesses of the existing methodology, as well as an evaluation of the strengths and weaknesses of new methods, techniques, and tools. While the general requirements for methodologies presented in Section 3 provide a framework for modification and evaluation of methodologies, more specific criteria for evaluation can also be identified. Unfortunately, few of these criteria are quantifiable. Furthermore, differences in organizational structures, software development practices, and system characteristics make it difficult to assign relative weights to the criteria. (Individual organizations, considering their own specific needs, may be able to do so, however.)

The criteria are divided into four major categories: technical, usage, management, and economic. Some of the allocations to categories is arbitrary. The ordering within categories has no significance.

5.1 Technical Characteristics

By technical characteristics we mean features that pertain to the support of various technical concepts by the method.

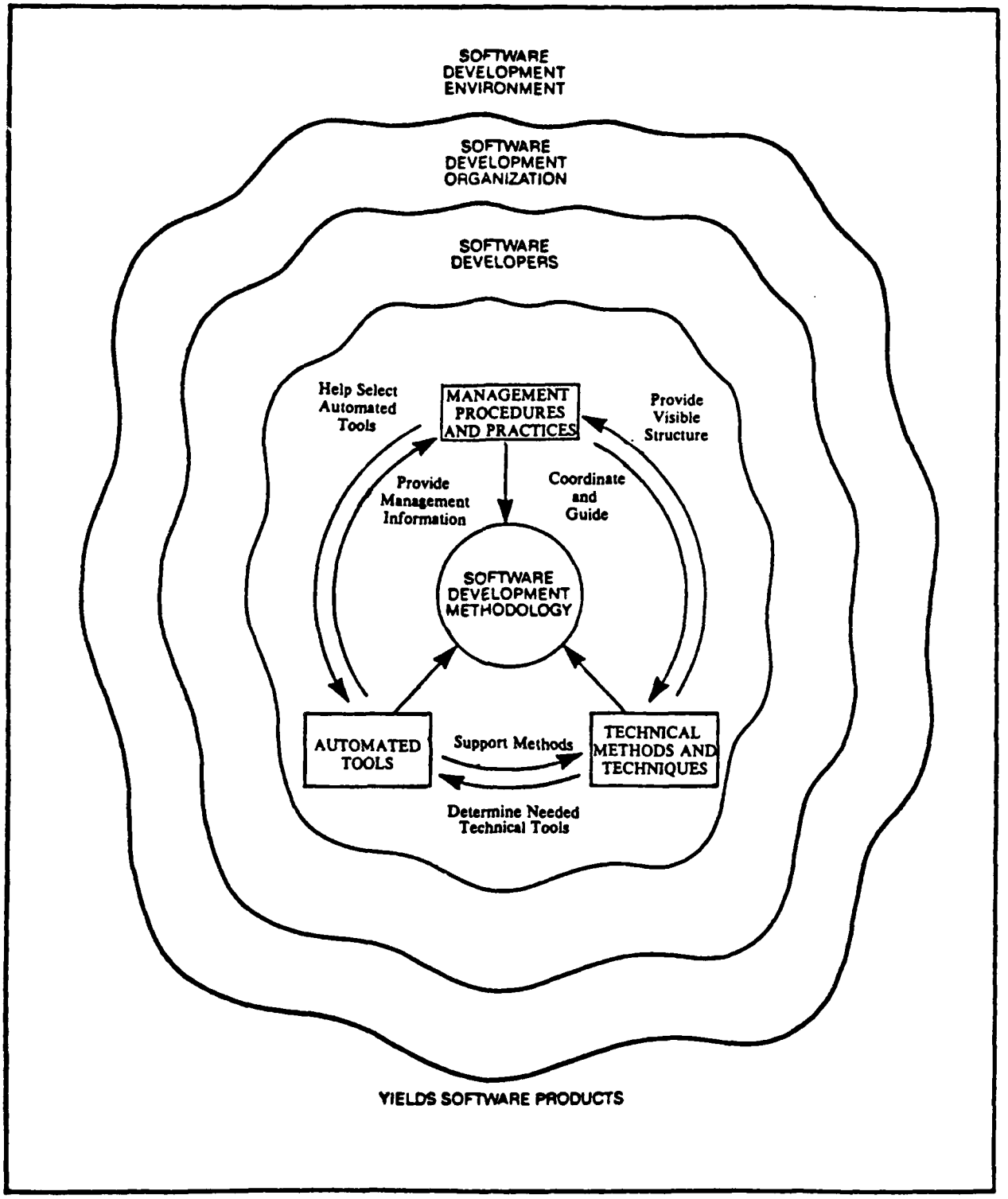


Figure 2 -- Software Development Environment

Function Hierarchy - A situation can be represented in which a function at one level is actually composed of several interconnected functions that exist at a level of greater detail.

Data Hierarchy - The concept of data classes; a situation can be represented in which a set of data at one level is actually composed of several interrelated pieces of data that exist at a level of greater detail.

Interfaces - The concept of having distinct and well-defined boundaries between processes or sets of data. Software systems and the large information systems of which they are a part should contain many distinct parts. Each part should have a clear definition so that it can be dealt with as a separable unit. If this is the case, then we have interfaces, or connections, between the various parts.

Control Flow - Representation of the sequence in which processes will take place.

Data Flow - Representation of the flow of information types between various processing elements and/or storage elements in the system.

Data Abstraction - The concept of hiding information about the implementation of a data type and providing a set of implementation-independent functions for use of the data type.

Procedural Abstraction - an algorithm for carrying out some operation is abstracted to a single name that can be used to invoke to procedure without knowing the details of its implementation. (Transactions are an instance of this case.)

Parallelism - a situation in which two or more cooperating sequential processes are concurrently in execution.

Safety - the avoidance of run-time failures which could lead to the loss of life or the occurrence of other catastrophic consequences.

Reliability - the absence of errors that lead to system failure.

Correctness - fidelity to functional specifications.

5.2 Usage Characteristics

Usage characteristics refer to those features relating to the methodology's application to development situations. These include:

Understandability - How easy it is for someone who is interested in the system being developed, but not especially knowledgeable in the technique, to understand the results of the development.

Transferability - The degree to which the method or tool can be successfully taught to personnel not previously familiar with it. This includes not only how easy it is to teach and learn, but also how well formulated it is.

Reusability - The ease with which previously created designs, code, or other work products can be reused in a new project.

Computer Support - Existence of automated tools which can be easily obtained and which aid in the use of the methodology or some of its steps.

Life-Cycle Range - The span of phases in the development life-cycle over which the tools or method can be usefully applied. This must by necessity be an approximate measure.

Task Range - The span of tasks to which the item may be usefully applied. Traditional classifications of software applications (business, scientific, systems, etc.) are not very useful for this evaluation. There are scientific and business problems that share the same problems while there are other tasks that say, within the business category, are quite dissimilar.

Cohesiveness - The extent to which the technical methods, management procedures, and automated tools may be combined to support the methodology.

Extent of Usage - A judgment as to how widespread the current usage of the methodology is.

Ease of phase transition - The extent to which information developed at one phase of development supports work to be done at a subsequent phase (e.g. from analysis to design).

Decision highlighting -- the ability of a technique to make visible and highlight key technical or project decisions (e.g. choice of data structures, nearness to completion of a project); the ability of a technique to illuminate the consequences of a development decision.

Validation - the extent to which the methodology assists in the determination of system correctness

Repeatability - the extent to which similar results are obtained when the methodology (or an included aspect) is applied more than once to the same problem.

Ease of change to work products - the amount of effort required to modify a work product when some aspect of requirements, specification, or design is changed.

5.3 Management Characteristics

These are the factors that relate to the ability of an aspect of the methodology to enhance the management of software development activities.

Manageability - The degree to which the method or tool permits standard management techniques of estimation, in-process status checking and control to be applied. The evaluations are based on the existence of well-defined steps and intermediate products which make management possible and the existence of management techniques applying specifically to the method at hand.

Teamwork - the extent to which the methodology and the development environment aid, rather than hinder, teamwork.

Phase Definitions - the identification within the methodology of development phases that represent intermediate stages of the development process.

Work products - the documents and systems that result from application of the methodology.

Configuration management - the way in which the methodology and/or its tools

provides for organization, tracking, and maintenance of the emerging work products, including control of releases and multiple versions.

Exit criteria - the way in which phases of development and work products are defined to provide explicit stopping or exit criteria for each stage of development.

Scheduling - the methodological support for project scheduling.

Cost estimation - the methodological support for cost estimation.

5.4 Economic Characteristics

Use of a software development methodology should also produce tangible economic benefits in software quality and the productivity of the software development organization, so that one should also consider the following aspects:

Local benefits (with respect to phase) - Compared to informal/traditional ways of carrying out the processes of a given phase, how much improvement can be expected through the use of a given tool or method?

Life cycle benefits - the benefit of a method relative to a particular stage, adjusted for the relative importance of that stage in the overall life cycle.

Cost of acquisition - the cost of obtaining training, tools, rights, etc. to use the methodology or some associated tool or technique.

Cost of use - the operational cost (computer time, forms, etc.) of using the methodology or some associated aspect.

Cost of management - the cost of managing the methodology or some associated technique.

6. CONCLUSION

The purpose of this document is to present the rationale for software development methodologies and to establish a framework for the description and creation of such methodologies in conjunction with embedded systems and Ada Programming Support Environments. To a lesser extent, we have tried to identify areas where additional research and development are needed before full methodological support becomes available.

There are many ways in which systems can be constructed and many tools that can be used. There are many existing methodologies and many more are likely to emerge based on different concepts of problem solving, approaches to system structuring, and organizational structures. The emergence of several widely used methods for analysis, specification, and design, combined with compatible APSE's, should lead to a small set of technical approaches to the specification, design, development, and validation of Ada programs, yielding standardized forms of work products.

Although the management aspects will differ among organizations, the narrowing of alternatives for technical methods and tools can contribute to enhanced maintainability of embedded systems. Subsequent work can recommend and/or develop technical methods that satisfy the framework described in this document. Such work can then aid in identifying tools that can be built in support of methodologically-oriented APSE's. The eventual integration of management procedures with suitable APSE's and Ada Methodologies can thereby further the goals set out when the Ada program was first undertaken.

Ada Methodology

Questionnaire Summary

Ada Methodology Questionnaire Summary

Maria Porcella
Peter Freeman

Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Anthony I. Wasserman

Medical Information Science
University of California, San Francisco
San Francisco, CA 94143

November, 1982

TABLE OF CONTENTS

1.	INTRODUCTION	2
2.	QUESTIONNAIRE DESCRIPTION	2
3.	SURVEY RESULTS	2
3.1.	SUMMARY OF THE RESULTS	3
3.2.	DETAILED DISCUSSION OF THE RESULTS	3
3.2.1	Key Concepts	5
3.2.2	Results by Questionnaire Subsection	5
	GENERAL METHODOLOGY ISSUES	5
	TECHNICAL ASPECTS	10
	Technical Concepts	10
	Workproducts	13
	Prescribed Workproducts	13
	Representation Schemes	13
	Ada Compatibility	19
	Quality Assurance	19
	AUTOMATED SUPPORT	19
	Automated Aspects	24
	Detailed Summary of Specialized Automated Support	24
	MANAGEMENT ASPECTS	24
	USAGE ASPECTS	24
	TRANSFERABILITY	30
4.	CRITIQUE OF THE SURVEY	30
5.	CONCLUSION	33
6.	APPENDIX 1: Identification of Methodology Developers	34
7.	APPENDIX 2: Original Questionnaire and Cover Letter	39

LIST OF TABLES

Table 1:	Methodologies Summarized	4
Table 2:	Key Concepts and Mechanisms	6
Table 3:	Software Development Life Cycle Coverage	9
Table 4:	Methodology Applicability	11
Table 5:	Technical Concepts Supported	12
Table 6:	Workproducts and Representation Schemes	14
Table 7:	Ada Compatibility	20
Table 8:	Quality Assurance Methods	21
Table 9:	Automated Aspects	25
Table 10:	Detailed Summary of Specialized Automated Support	27
Table 11:	Management Aspects	29
Table 12:	Usage Aspects by Methodology	31
Table 13:	Transferability Aspects	32

1. INTRODUCTION

In Spring, 1982, we developed a questionnaire on software development methodologies, identified 48 methodologies to be surveyed, and sent the survey to the developers of these methodologies. The intent of the survey was to gather conceptual, technical, and usage data of a general nature and, more specifically, to relate the data to potential methodology usage in an Ada development environment, wherever possible. The intent was not to describe any particular methodology in detail. The results of the survey are presented here in a form that captures the factual results and suggests a first level of generalization of the state of the art in software development methodology.

A questionnaire was chosen as the instrument for conducting the survey because resource constraints prohibited a more thorough survey method, such as interviewing. The questionnaire contained a combination of free-form and multiple-choice questions. This allowed respondents to describe the methodology in their own terms, as well as in standardized terms.

We tried to identify as many methodologies as possible, but we are certain that some were overlooked. Also, there are other methodological aspects that remain unaddressed. We believe, however, that our coverage of the field and our interpretation of the responses conveys an accurate representation of each approach and the state of the art. In this report, we assume that the reader is familiar with concepts of software development methodologies. We, of course, take full responsibility for our conclusions and interpretations.

2. QUESTIONNAIRE DESCRIPTION

The questionnaire (reproduced as Appendix 2) consisted of several parts:

(1) *Identification* - Asked for administrative information about the methodology, its developer, the preparer of the questionnaire, and the primary contact for future queries.

(2) *General Methodology Issues* - Sought broadly descriptive data about the key concepts of the methodology, its coverage of the software development life cycle and its perceived suitability in specific application domains.

(3) *Technical Aspects* - Explored the technical procedures used in the methodology and the mechanisms for representing some important technical aspects such as the visible workproducts, quality assurance procedures, and points of Ada compatibility. By "workproducts" we mean the documents and system products (internal and deliverable) that result from application of the methodology.

(4) *Automated Support* - Asked the respondent to describe the specialized automated tool support for the methodology, to ascertain its availability/portability, and to define the hardware environment needed for that support.

(5) *Management Aspects* -Delved into the methodology's coverage of the project, product, and people management aspects of software development. In particular, the product management dimensions of configuration control and evolution were explored because of their importance to integrated programming environments.

(6) *Usage Aspects* - Sought information about the experience with using the methodology in organizations, including the required effort for introducing the methodology, and the extent of projects and organizations involved.

(7) *Transferability* - Explored some technology transfer issues including the available transfer mechanisms and problems of introduction of new techniques into organizations.

3. SURVEY RESULTS

Of the 48 people/organizations who received the questionnaire, 29 responded (60 percent). However, two of the responses were apologies for not being able to answer the questionnaire. Appendix 1 lists the organizations associated with all the methodologies for which we received a positive response (27 in all). This summary only includes details on 24 of the responses, however. Three methodologies (DAISEE, DCDSM, and SSDM) were not included because they are still in the experimental stage and have not been used on any projects.

We have made little effort to evaluate the claims of the respondents. While we believe that all of the responses were made in good faith, the reader is cautioned to interpret "a methodology supports..." to mean "a methodology *claims* to support....".

Considering the length of the survey, we regard the response as excellent and extend our thanks to all of the respondents. We feel that the response indicates a growing interest in exploring software development methodologies.

3.1 SUMMARY OF THE RESULTS

The survey responses provide us with a good picture of the current stock of methodologies on which we can base some initial generalizations. The generalizations fall into the following categories: life cycle coverage, software development process, automated support, and management support.

Life Cycle Coverage

The methodologies may be grouped into three categories according to their life cycle coverage: those that cover the initial phases of the life cycle, up to design; those that cover from the initial phases up to implementation; and, those that attempt to control the life cycle of the product through its evolution.

Software Development Process

An second generalization we can make is that there is value placed on certain processes involved in software development: modelling activities, consistency checking procedures, formalized procedures for specification verification and validation, a trend toward the use of formal specification languages, use of graphical/diagrammatic representation schemes, as opposed to narrative, to capture systems documentation, management procedures, functions, and data.

Automated Support

There is also a general trend toward development and use of automated tools to support the methodology in the areas of document preparation, computer aided design and modelling, prototyping, code generation, database/data dictionary support, etc. The methodologies may be placed into four basic categories for this purpose:

1) those without automated support and no plans for such support;
2) those that plan to develop automated aids, but which presently have none;
3) those providing tools in a general support environment that have been created independently of the methodology;
4) those where there is a tight coupling with the tool environment (specialized tool support for specific aspects of the methodology).

Management Support

As far as management issues are concerned, the methodologies surveyed differ substantially on whether they directly provide management guidance, and on what type of guidance they provide. Most methodologies are sensitive to some combination of project, technical, and people management issues; but only slightly more than one-third attempt to provide guidance on all three dimensions.

3.2 DETAILED DISCUSSION OF THE RESULTS

Our detailed discussion first present key concepts, then the results by questionnaire subsection. Table 1 lists the 24 methodologies (with the acronyms for each) used for this analysis. If the developer did not provide an acronym for the methodology, we created one and use that acronym throughout this report. No significance should be attached to these assigned names beyond this report.

METHODOLOGIES SUMMARIZED	
Methodology Mnemonic	Full Name of Methodology
ACM/PCM	Active and Passive Component Modelling
DADES	Data Oriented Design
DSSAD	Data Structured Systems Analysis and Design
DSSD	Data Structured Systems Development
EDM	Evolutionary Design Methodology
GEIS	Gradual Evolution of Information Systems
HOS	Higher Order Software
IBMFSD-SEP	Adaptation of IBM Federal Systems Division Software Engineering Practices
IESM	Information Engineering Specification Method
ISAC	Information Systems Work and Analysis of Changes
JSD	Jackson System Development
MERISE	
NIAM	Nijssen's Information Analysis Method
PRADOS	Projektabwicklungs- und Dokumentationssystem
REMORA	
SADT	Structured Analysis & Design Technique
SARA	System ARchitect's Apprentice
SD	System Developer
SA-SD	Structured Analysis and Structured Design
SDM	System Development Methodology
SEPN	Software Engineering Procedures Notebook
SREM	Software Requirements Engineering Methodology
STRADIS	STRuctured Analysis, Design and Implementation of Information Systems
USE	User Software Engineering

Table 1: Methodologies Summarized

3.2.1 Key Concepts

Although a subsection of the questionnaire asked specifically for the key concepts or principles underlying the methodology, key concepts were discussed indirectly in many of the free-form questions. We have tried to extract the general ideas of each methodology, wherever stated, and have gathered them in this section. The key concepts may be realized in practice by diverse mechanisms -- for example, "bottom-up design" and "top-down design" are two fundamentally different practices for achieving the goal of "improved manageability of software production".

In constructing Table 2, we first collected the conceptual information contained in the set of questionnaires. We extracted this information verbatim, in most cases. We then analyzed the results for some logical presentation. The key concepts fell into two major categories: guiding methodological concepts, and specific mechanisms for achieving software production. Each of those categories was further organized into broad classes: general, automation, technical, management, and user relations. Table 2 is then a synthesis of the responses, using the respondents' terms insofar as possible, and combining common responses.

NOTE: The individual concepts and mechanisms are listed without any specific attempts at lateral correspondence between the two columns.

3.2.2 Results by Questionnaire Subsection

The following discussion presents the responses by subsections of the questionnaire. The major subsections of the questionnaire were as follows: identification, general methodology issues, technical aspects, automated support, management aspects, usage aspects, and transferability.

We have organized the data into tables where appropriate, and reported it in a more informal way if there were no obvious dimensions for comparison.

GENERAL METHODOLOGY ISSUES

The following subsections summarize the responses on software development life cycle coverage and applicability.

Software Development Life Cycle Coverage

The data on life cycle coverage (from question 2) is summarized in Table 3, where we attempted to match the life cycle phases described by the respondents to the life cycle phases outlined in "Ada Methodologies: Concepts and Requirements," (AJPO, November, 1982). We have analyzed the responses beyond the simple answers given to question 2, drawing inferences from other sections of the questionnaire. Often the respondents just listed phases covered, and misunderstandings concerning the meaning of terms made it necessary for us to delve further into the questionnaire.

Table 3 shows that nearly all methodologies support "Architectural" Design and Functional Specification (23 & 24 of 24 respectively). We emphasize "Architectural" because "Detailed" Design cannot be included in these figures. Most respondents did not refer to "Detailed Design" specifically. "Design" was the most common response; and we judged that to be too general.

Seven-eighths (21 of 24) provide support for the implementation level. More than half (14 & 13 of 24, respectively) support requirements analysis and/or evolution, and more than half (14 of 24) have a distinct validation phase.

Another way of viewing these results is that about one-sixth (4 of 24) of the methodologies provide support only through the architectural design stage; another seven provide support only through implementation and/or validation; and the rest consider post-construction and evolution. Thus, the methodologies fall into 3 basic categories along this dimension, as shown below:

KEY METHODOLOGICAL CONCEPTS & MECHANISMS	
Guiding Methodological Concepts for Software Production	Specific Mechanisms for Achieving Software Production
GENERAL GOALS: Simplicity Balance between simplicity and complexity Control of complexity Rigor Apply to any problem domain	*GENERAL FEATURES:* Use of "state-of-the-art" methods, tools, and management techniques Proven procedures (based on experience) Provide a software development model (scenario) Improve the quality of software products Systematic, step-by-step design
AUTOMATION GOALS: Automate life cycle processes that are conventionally done manually and redundantly Integrated family of tools Provide graphical tools	*AUTOMATED PROCESSES:* Simulation of man-machine dialogues Rapid prototyping Transform user requirements into Functional Specification Specification library Verify Functional Specification Transform specifications into implementation Automatic programming Automatic documentation Report generation Test generation Consistency checking

Table 2: Key Concepts & Mechanisms

KEY METHODOLOGICAL CONCEPTS & MECHANISMS	
Guiding Methodological Concepts for Software Production	Specific Mechanisms for Achieving Software Production
	TECHNICAL FEATURES:
	Implementation-independent design
	Logically guaranteed requirements definition
	Explicit specification & modelling of the *environment* in which a system will exist
TECHNICAL GOALS:	Data (entity) modelling
	Event (activity) modelling
Criteria given for all technical aspects	Integration of time representation and logical data model
Formalization of specifications and design	
Verification of specification and design decisions	Provide a language for recording specification and design decisions
Provide an explicit model of the real world	Algebra of structures (set-to-set relationships)
Overall optimization of logical/physical data and processing architectures	Graphical representation
	Information hiding
Support design of concurrent hardware and software systems	Modularity
	Module interface design
	User interface (dialog) design
	TECHNICAL STRATEGIES:
	Concentration on system outputs
	Specific sequencing of data modelling and functional decomposition
	Parallel (independent) approach to modelling data and process
	Simulation
	Prototyping
	Top-down design
	Bottom-up design
	Bottom-up implementation
	Top-down testing

Table 2: Key Concepts & Mechanisms
(continued)

KEY METHODOLOGICAL CONCEPTS & MECHANISMS	
Guiding Methodological Concepts for Software Production	Specific Mechanisms for Achieving Software Production
PRODUCT MANAGEMENT GOALS: Provide quality control Provide version control Provide configuration management Provide an explicit model of the software development process *PROJECT MANAGEMENT GOALS:* Improve the manageability of software production Improve the efficiency of software production Improve the SE practices of programmers in the organization 100% centrally verified consistency Improve productivity over the entire life cycle	*MANAGEMENT PRACTICES:* Use central knowledge base Use existing software "Teamwork" working environment Role definitions Evolutionary planning of tasks Workproduct definitions Workproducts on appropriate forms Validated workproducts Author/Reader Cycle Wide distribution for review Workproduct reviews by author, team, and management Provide estimation and scheduling techniques
USER/DEVELOPER COMMUNICATION: Balance between user involvement and systematic software development User envolvement in requirements definition	*USER INVOLVEMENT PRACTICES:* Formal validation of models and specifications by user Formal customer acceptance testing User orientation & control of the development process

Table 2: Key Concepts & Mechanisms
(continued)

SOFTWARE DEVELOPMENT LIFE CYCLE COVERAGE						
Methodology	Req'mnts Analysis	Functional Specification	Design	Implementation	Validation	Evolution
ACM/PCM	x	x	x	x		x
DADES	ISAC*	x	x#			
DSSD	x	x	x	x	x	x
DSSAD	x		x	x		
EDM	x	x	x	x	x	x
GEIS	x	x	x	x		
HOS		x	x	x	x	x
IBMFSD-SEP			x	x	x	
IESM		x	x	x		x
ISAC	x	x	x			
JSD	x	x	x	x		
MERISE	x	x	x	x	x	x
NIAM		x	x	x	x	x
PRADOS	x	x	x	x	x	x
REMORA		x	x	x	x	
SADT	x	x	x			
SARA	x	x	x	x	x	
SA-SD	x	x	x	x	x	x
SD		x	x	x		
SDM		x	x	x	x	x
SEPN	x	x	x	x	x	
SREM	DCDSM*%	x	x	x	x	x
STRADIS	x	x	x	x	x	x
USE		x	x	x		x

Table 3: Software Development Life Cycle Coverage

Key:

- x = phase covered by the methodology
- * = methodology is coupled with this other methodology, tool, method, etc.
- $ = partial coverage
- # = does not cover detailed design
- % = in developmental stages
- blank = phase not covered

Through Design	Through Implementation and/or Validation	Through Evolution
DADES	DSSAD	ACM/PCM*
ISAC	GEIS	DSSD*
SADT	JSD	EDM*
SD	REMORA	HOS
	SARA	IESM
	IBM-FSD	MERISE*
	SEPN	NIAM
		PRADOS*
		SA-SD*
		SDM
		SREM*
		STRADIS*
		USE

The methodologies under the "Through Evolution" column that are flagged by an asterisk ("*") span the entire life cycle. However, they may not support *each* phase of the life cycle. For example, some do not have a distinct requirements analysis phase; others do not distinguish various intermediate phases.

While support for the entire life cycle is essential for a software development methodologies, there are few current methodologies that fulfill that requirement. Since most of the methodologies do not support the entire life cycle, we would have expected more of the respondents to describe how their methodologies could be used with other techniques to cover the entire life cycle. However, this was not done, perhaps due to the way the question was interpreted. "A *system* for developing systems" may not have suggested "complete life cycle coverage" to the respondent despite our intent.

Methodology Applicability

The responses on applicability (Table 4) come from questions 4 & 5 of this portion of the questionnaire. Both questions were objective, with explicit instructions on how to respond. Even so, some responses answered question 4 "yes" or "no" when we asked for more specific suitability categories; hence, some entries in the table are "x" where they ought to be "W", "S", etc.

The methodologies surveyed are *most* suitable for data processing and database systems (21 of 24) and *least* suitable for expert systems and artificial intelligence applications (4 of 24). Approximately half of them (13 of 24) are well-suited to embedded, real-time systems. Moreover, only a few (HOS, SREM, and with a slight exception, SADT) are well-suited for all application domains. Also, while most (15 of 24) are intended for use on systems of all sizes, some methodologies (7 of 24) may not be appropriate or affordable for small systems and a small number of methodologies (2 of 24) may not be appropriate for large systems.

Suitability of a particular methodology, or set of methodologies, for an Ada environment will depend partly on the needs of the development organization, such as its concerns for a range of applications and project size. The variable applicability reported suggests that a software development organization will need to acquire some expertise in evaluating methodologies with respect to its specific organizational needs.

TECHNICAL ASPECTS

The following subsections report the data gathered on technical concepts, workproducts and representation schemes, compatibility with Ada programming features, and quality assurance techniques.

Technical Concepts

Table 5 summarizes methodology support for certain technical concepts: function hierarchy/decomposition, data hierarchy/decomposition, interface definitions, data flow, sequential control flow, parallelism, and formal program verification. The question was answered in many ways, which accounts for the simple reporting scheme. We had asked "how" the concept was supported, and received answers like "Not at all", "Satisfactorily", and "via data flow diagrams". A combination of the two latter type responses was what we had sought.

One-third (8 of 24) of the methodologies support all the technical concepts we identified. When formal program verification and concurrency (which are the least supported, 13 & 17 of 24 respectively) are ignored, the presence of "complete" technical support increases substantially. In that case, three-fourths

METHODOLOGY APPLICABILITY									
Methodology	Application Type						Size		
	Embedded	Sci/Eng	O/S	Tools	DP/DB	Expert/AI	Sm	Med	Lg
ACM/PCM	I	I	I	S	W	S	...	x	x
DADES	S	I	I	S	W	I	...	x	x
DSSAD	W	W	S	S	W	S	x	x	x
DSSD	W	W	S	W	W	IE	x	x	x
EDM	W	W	IE	W	W	IE	x	x	x
GEIS	I	I	W	W	W	I	x	x	x
HOS	W	W	W	W	W	W	x	x	x
IBMFSD-SEP	x	x			x		...	x	x
IESM	S	I	S	W	W	I	x	x	x
ISAC	I	W	S	W	W	I	x	x	x
JSD	W	S/W	W	W	W	IE	x	x	x
MERISE	W	S	S	S	W	IE	x	x	x
NIAM	S	S	S	W	W	W	x	x	x
PRADOS	S	W	S	W	W	I	x	x	x
REMORA	W	S	I	W	W	I	x	x	x
SADT	W	W	W	W	W-S	W	x	x	x
SARA	W	W	S	S	S	I		x	x
SA-SD	W	W	W	W	W	I	x	x	x
SD		S	S	S	W	S	x	x	...
SDM	S	S	W	W	W	S	x	x	x
SEPN	W	S	I	S	S	I	...	x	x$
SREM	W	W	W	W	W	W	...	x	x
STRADIS	W	W	W	W	W	...	x	x	x
USE	I	W	I	W	W	S	x	x	IE

Table 4: Methodology Applicability

Key:

W = well suited
S = satisfactory
I = inappropriate
x = suitable
(W or S not specified)
IE = insufficient experience
... = no answer
$ = "Large is 250K lines, 75-100 effort years"
blank = is not suitable

Sci/Eng = Scientific/Engineering
O/S = Operating Systems
DP/DB = Data processing, database
AI = Artificial intelligence

TECHNICAL CONCEPTS SUPPORTED							
Methodology	Function Hierarchy/ Decomp.	Data Hierarchy/ Decomp.	Interface Definitions	Data Flow	Seq. Control Flow	Concur- rency/ Parallelism	Formal Program Verification
ACM/PCM	x	x	x	x	x	x	x
DADES		x	x				x
DSSAD		x		x	x		
DSSD	x	x	x	x	x	x	
EDM	x	x	x	x	x	x	
GEIS	x	x	x				
HOS	x	x	x	x	x	x	x
IBMFSD-SEP	x	x	x	x	x	x	x
IESM	x	x	x	x	x	x	x
ISAC	x	x	x	x	x	x	
JSD		x	x	x	x	x	
MERISE	x	x	x	x	x	x	
NIAM	x	x	x	x	x		x
PRADOS	x	x	x	x	x		
REMORA	x	x	x		x	x	x
SADT	x	x	x	x	x	x	x
SARA	x	x	x	x	x	x	x
SA-SD	x	x	x	x	x	x	
SD	x	x		x			x
SDM	x	x	x	x	x		x
SEPN	x	x	x	x	x	x	
SREM	x	x	x	x	x	x	x
STRADIS	x	x	x	x	x	x	
USE	x	x	x	x	x	x	x

Table 5: Technical Concepts Supported

Key:

x = *concept supported by the methodology*
blank = concept not addressed

(18 of 24) of the methodologies in our study support those concepts.

This result is open to several interpretations:

(1) our list of technical concepts may exceed the set that is necessary and sufficient for a methodology;

(2) few methodologies provide comprehensive technical support for problem solution and system development;

(3) users of the methodologies implicitly use these concepts, but haven't recognized them explicitly.

We believe that there is some truth to each of these interpretations.

Workproducts

Table 6 lists the prescribed workproducts and the representation schemes used by each methodology. By "representation schemes" we mean the graphic and textual notations (e.g. Petri nets, data flow diagrams, formal specification language) by which the relevant information is captured and communicated.

We have made little attempt to classify the responses because we wanted to avoid, as much as possible, misrepresenting the data; moreover, specialized developer terminology, coupled with sparse information, precluded such analysis. Hence, most of the table entries use the terminology provided by the respondent.

However, we have tried to draw some conclusions about the workproducts and representation schemes used. One observation is that, "prescribed workproducts" and "representation schemes used" were often interpreted as the same "objects". This was not anticipated, since our intent in asking the two questions was first to identify "what" was the *organized output* of each phase, and then to identify specifics on "how" that output was captured, or what various forms that information took.

The discussion of the results of this subsection is divided into two parts: prescribed workproducts and representation schemes.

PRESCRIBED WORKPRODUCTS

The most frequently prescribed workproducts were Specifications and Designs. In addition, a frequently mentioned "workproduct" was a library or data dictionary. In addition, methodologies having database support (DSSD, GEIS, NIAM, SARA, and SREM) mentioned the capability of producing some number of reports (what we would call "workproducts") that were not specifically identified.

For most of the methodologies, workproducts identified were those that were produced up to but not including the code. Therefore, we had to infer "code" from other information about the methodology (e.g. support through implementation). A few methodologies (SARA, SEPN, and STRADIS) mentioned explicit test environment workproducts. Only STRADIS and IBM/FSD-SEP explicitly mentioned post-construction workproducts.

These are interesting results relative to the life cycle coverage (Table 3). The implementation phase, for example, is shown as being supported extensively, despite no *explicit* methodological guidelines for workproduct production. It appears that prescribed workproduct definitions and phases of direct life cycle coverage do not have a one-to-one correspondence; hence, it seems that workproducts are not well-integrated into methodologies. Lastly, only STRADIS specifically mentioned *management workproducts* although many other methodologies have them in some form (e.g. project models, schedules, etc.).

REPRESENTATION SCHEMES

Representation schemes generally, take the form of data flow diagrams, structure charts, state machine diagrams, entity-relationship diagrams, other types of graphs, program design language (PDL), and data dictionaries. Many of the methodologies with database support mentioned some formal specification language as the mechanism for capturing that data. In general, the objects of the modelling process are data, activities or operations, and their relations. (Note: It was not clear from many of the responses whether "time" was featured as a subject to model.)

What is noteworthy is the emphasis on *graphical* representation schemes and predicate-calculus-based specification languages, which lend themselves to machine processing. Moreover, the two schemes

WORKPRODUCTS & REPRESENTATION SCHEMES		
Methodology	Prescribed Workproducts	Representation Schemes Used
ACM/PCM	Specifications Object & Action schemes and skeletons Hierarchy of data abstractions and operations	Graphical: for data & operations Predicate logic: Functional Specifications
DADES	Formal Specifications Architectural Design	Data flow: Architectural Design Narrative Tables: requirements Formal spec language Diagrams: describe sub-systems, process structure
DSSAD	Jackson-type data structure diagrams Interaction diagrams	Jackson-type data structure diagrams Interaction diagrams
DSSD	Structured Requirements Definition Structured System/Data Base Design (Library) Structured Program Design	Functional data flow diagrams Entity (contextual) diagrams Assembly line diagrams Input/Output Diagrams Event structures
EDM	Standardized sheets for: Requirements analysis Specification Design (all other tasks)	Data flow diagrams Hierarchical charts Data structure tables Relational schemes control flow diagrams Interface design
GEIS	Specifications Library (of Specs)	Chart: function/data relationships
HOS	Formal Specifications: (in a library) Graphical control maps Program code	AXES spec language: data type axioms, primitive operations, control structures, and other formal mechanisms Graphical control maps Functional hierarchies

Table 6: Workproducts & Representation Schemes

WORKPRODUCTS & REPRESENTATION SCHEMES		
Methodology	Prescribed Workproducts	Representation Schemes Used
IBMFSD-SEP	Workproducts for each software engineering practice (SEP) Design Programs: (in configured controlled libraries)	State machine diagrams Structure charts Program design language (PDL)
IESM	All workproducts defined by the METHOD System Specification: measurable objectives of the system	Sentences: formal System Specification Axioms: (defined by the METHOD) Data flow diagrams
ISAC	Activity model of current situation Activity model of chosen change alternative Change plan Activity model with information subsystems Priority plan for information subsystems Detailed information analysis models Equipment-independent data system model Equipment-adapted data system model	Activity graphs (A-graphs) Information flow graphs (I-graphs) Component graphs (C-graphs) Process tables (P-tables) Data system design graphs (D-graphs)
JSD	Entity & Action lists Entity Structures (Trees) System Specification Structure Texts (like attribute grammars) System implementation diagrams Executable text (JCL, programming language, etc.) Database design	Tree Structures (diagrams & texts) Data flow diagrams Database diagrams (optional)

Table 6: Workproducts & Representation Schemes
(Continued)

WORKPRODUCTS & REPRESENTATION SCHEMES		
Methodology	Prescribed Workproducts	Representation Schemes Used
MERISE	Standard set of documents at each definite phase: planning, problem spec, design, verification, simulation, coding, prototype, construction, commissioning, launching and maintenance	Process diagrams (Petri nets) Entity-relationship diagrams Bachman diagrams Data flow diagrams Programming trees
NIAM	Knowledge Base: Integrated software information system capable of generating system documentation, cross-refs, and other reports Process descriptions	Decomposition schemes Information flow diagrams Information structure diagrams Information dictionary Formal spec language: process descriptions
PRADOS	Requirements Solution Concept Designs Code	SADT diagrams: analysis DSA diagrams (Datenstruktur analyse) (based on entity-relationship model) Data structure charts: design module specifications Nassi-Shneiderman diagrams: (programming)
REMORA	Conceptual Schema: relations of static and dynamic aspects Logical Schema: logical data schema and transactions Synchronization schema Code	Formal spec language: conceptual schema Diagrams: 3-alternate graph conceptual schema Logical data graph Transactions sequencing graph
SADT	Models (composed of diagrams) Kits: for review and approval procedure Node indexes and trees Large schematic diagrams algorithmically derived from single or multiple models	Diagramming language: Activity diagrams Data diagrams State diagrams Text Glossary Node Index (Table of Contents) Schematics (for walk-throughs)

Table 6: Workproducts & Representation Schemes (Continued)

WORKPRODUCTS & REPRESENTATION SCHEMES		
Methodology	Prescribed Workproducts	Representation Schemes Used
SARA	Requirements document Design models: structural (SL1), behavioral (Graph Model of Behavior), Module Interface Description Reports: from analysis and Test Environment QA requirements document Detailed design specification Evaluation transcripts	Represent structure: SL1 (modules, sockets, interconnections) Represent behavior: GMB (control flow, data flow, interpretation domains) MIL (Ada specification parts)
SA-SD	Analysis: Structured specification Data Dictionary Mini-specification State Transition Model Design: Design specification Database Design Operational constraints Physical constraints Implementation: structured code	Data Flow Diagrams Structure charts Data structure diagrams Finite state diagrams Decision tables Program design language (PDL)
SD	Concept structure: (requirements spec) Database Design Conceptual schema Access path specifications Algorithm specifications Code	Graphical (with additional texts) Data flow diagrams Concept structures
SDM	Over 100 forms: Each task within each activity within each phase prescribes one or more products	"All such [axioms, DFD's, finite state diagrams] design techniques fit..."
SEPN	"Each SEPN describes a specific work product and each is defined in terms of standard symbology, forms, or criteria to be met."	"Each SEPN contains a recommended standard format."

Table 6: Workproducts & Representation Schemes (Continued)

WORKPRODUCTS & REPRESENTATION SCHEMES		
Methodology	Prescribed Workproducts	Representation Schemes Used
SREM	Requirements Definition Software Requirements Specification Documentation from queries to requirements database	Stimulus-response R_NET (based on graph-model theory) Formal requirements statement language Relational database description
STRADIS	Initial Study Report Detailed Study Report Draft Requirements Statement Outline Physical Design Total Requirements Statement Design Statement Accepted But Uninstalled System Installed System Documentation Package Management forms: for planning and control	Data Flow Diagrams (requirements) Static & Dynamic Data Models Design Data Flow Diagrams (transition between Analysis and Design) System structure charts: "sub system" design PDL: program design
USE	Specification: Dialogue design Database design Architectural Design: Structure charts Detailed design Prototype Source code (in PLAIN)	Data Flow Diagrams Data models (E-R models, semantic hierarchy models) Augmented Transition Diagrams Structure charts (Architectural design) Program design language (PDL) First-order predicate-calculus (behavioral abstraction)

Table 6: Workproducts & Representation Schemes
(Continued)

often exist together, in complementary relationship. For example, the information needed to perform automated consistency checking will often need to be captured in a machine-processable form, as well as in graphical or textual form for human review and understanding.

Ada Compatibility

Table 7 summarizes the response to question 4, which was aimed at uncovering how well the methodology supported design to a detailed level, and how well that design mapped into specific Ada features. In general, the question was answered in a straightforward fashion. However, there seemed to be some confusion over *the intent to use Ada as a design language.* Several respondents concluded that we were asking the question from that point of view. Hence, their answers addressed the appropriateness of Ada as a design language.

Also, there was an objection to the question on the grounds of implementation-independence -- a defensive reaction that might have been anticipated. However, considering the specificity of this survey to Ada development environments, the issue cannot be ignored.

Of the total number of responses for this question (24x6=144), 24 were answers of "not known". They are represented in the table by "?". In addition to those, another 30 items (represented by "...") were not answered in "yes", "no", or "not known" terms. When combined, they represent more than one-third of the answers in the table (54 of 144). That set of "responses" intersects 18 of 24 methodologies, indicating that many of the methodology developers have yet to give attention to the mappings between their technical representations and the Ada language. This would have to be done before we could determine the real mix of support for detailed design and implementation in Ada.

If we look at the four Ada constructs (packages, tasks, generics, and exception handling) we find that only one-fourth (6 of 24) of the methodologies claim to support them all. Moreover, while it appears that there is ample power in those methodologies for mapping into Ada constructs, it is not clear how direct the mappings are. Many responses had a flavor of conjecture, e.g., "*could* be expressed by...". The methodological representations may have the potential to express certain detailed design aspects, without explicitly demanding them. In that case, the mappings would only be obvious to very knowledgeable designers and programmers. Thus, although most methodologies support detailed design into Ada, designers may need extensive training and experience with Ada to develop their intuition regarding the "practicality" of implementing the design in Ada.

While only a few (4 of 24) methodology developers stated that there are no serious incompatibilities between their methodology and Ada, most do not know (or ignored the question). Three, however, pointed to incompatibilities between Ada and database/ data modeling applications. Again, though, the response indicates limited familiarity with Ada on the part of the methodology developers.

Quality Assurance

Table 8 summarizes the findings on quality assurance and validation measures prescribed by each methodology. Again, the table simply records the responses in slightly edited form.

In general, all methodologies appear to apply some quality assurance techniques to their workproducts, or outputs. Author/reader cycle, reviews, structured walkthroughs, design and code inspections, were the most common forms of quality assurance techniques mentioned. Validation procedures against the completed system are also common, however, differing as to when they occur. Either the system is validated be the developer before release by consistency checking, simulation, and other forms of internal testing, or it is subject to acceptance tests by the user upon delivery.

Some methodologies include automated quality assurance and validation techniques for consistency checking and other forms of testing. However, for the most part the techniques are performed manually.

AUTOMATED SUPPORT

The questions on automated support address existence of tools and the range of environments in which those tools can be used. We also attempted to determine the extent to which use of the tools is *required* with the methodology, and the future plans of the developer to provide additional tool support.

Flexibility of the tools was also a consideration. Consider a set of automated aids that are tightly coupled to a methodology. The tight coupling may give the methodology a "well-integrated" rating, but

ADA COMPATIBILITY						
Methodology	Ada Construction				Machine Representation	Serious Incompatibility?
	Packages	Tasks	Generics	Exception Handling		
ACM/PCM	yes	no	yes	yes	no	Database%
DADES	?	?	?	?	?	?
DSSAD	...	...	...	...	...	...
DSSD	yes	yes	?	yes	?	?
EDM	?	?	?	?	?	?
GEIS	no	no	no	no	no	...
HOS	yes	yes	yes	yes	yes	no
IBMFSD-SEP	yes	?	yes	yes	yes	...
IESM	yes	yes	yes	yes	no	?
ISAC	yes	yes	...	no	yes	no
JSD	yes	?	no	?	?	?
MERISE	yes	yes	yes	yes	no	...
NIAM	yes	yes	?	?	?	...
PRADOS	...	...	...	...	...	...
REMORA	yes	yes	yes	possible	no	no
SADT	yes	yes	yes	yes	yes	no
SARA	yes	yes	yes	yes	no	...
SA-SD	yes	yes	...	yes	...	Database&
SD	yes	yes	yes	...	...	...
SDM	yes	yes	...	yes	yes	...
SEPN	no	no	no	no	no	yes
SREM	yes	yes	yes	yes	no	yes
STRADIS	...	...	...	...	...	?
USE	yes	yes	no	yes	no	Database&

Table 7: Ada Compatibility

Key:

? = "Not known"
... = no answer
yes = methodology supports mapping into Ada feature
no = methodology does not support mapping into Ada feature
% = Query language
& = Modelling/Design; I/O

QUALITY ASSURANCE METHODS		
Methodology	QA methods applied to workproducts	How the completed system is validated against the original requirements
ACM/PCM	Author/reader cycle: (Object/Action Schemes & Skeletons) Testing/Formal verification: Specifications	Formal verification (partial) Testing the prototype
DADES	Formal validation: Specifications Architectural design	Consistency/Derivability analysis
DSSAD	Structured walkthroughs	"User interaction at each stage of the design to confirm correctness"
DSSD	Structured walkthroughs Formal testing techniques: design, test	System outputs validated against output requirements Physical requirements (volumes, frequencies) checked Calculation rules validated
EDM	Team review Review by author Management review Computer-aided testing: machine-readable workproducts	Same as QA for workproducts.
GEIS	Author/reader cycle	End-user feedback
HOS	Automated analyzer: Specification Resource allocation tool (RAT): Code guaranteed correct	"Validation done DURING development through use of automated tools at each phase. Final 'over-all' validation is superfluous."
IBMFSD-SEP	Design & code inspections Testing: (unit & product)	Hierarchical software test Product specification testing MIL standard specification testing
IESM	Consistency checks: Specification statements Documentation review Structured walkthrough Small implementation steps Test data generation Automated testing	Automated tests "The original requirements" are well specified in the system specification."
ISAC	Structured walkthroughs Inspections	"Documentation in property tables of how the original requirements are fulfilled" Prototype experiments

Table 8: Quality Assurance Methods

QUALITY ASSURANCE METHODS		
Methodology	QA methods applied to workproducts	How the completed system is validated against the original requirements
JSD	Author/reader cycle Structured walkthroughs Inspections	Manual checking (transformation of specification to implementation)
MERISE	Record of designer/user dialog Formal design inspection Formal user validation of models and specifications Prototype development Author/reader cycle Structured walkthroughs Inspections Benchmarks and testing	Internal test runs & benchmarks Formal user test runs Operation center & maintenance team testing
NIAM	Formal verification of information flow diagrams (IFD's) and constraint definitions Analyze constraints Walkthroughs Impact-mechanism (shows impact of change in specifications)	Acceptance test
PRADOS	Author/reader cycle: SADT diagrams Structured walkthroughs: designs Inspections: code	"No explicit methods!"
REMORA	Automated consistency checks: conceptual schema Simulation of conceptual schema Alternative logical solution evaluation	By simulation
SADT	Author/reader cycle Structured walkthroughs Some automatic consistency checking "Graphical notation forms a rigorous formal language."	Cross-referencing: from notation on diagrams Walkthrough sessions with the user/specifier

Table 8: Quality Assurance Methods
(Continued)

QUALITY ASSURANCE METHODS		
Methodology	QA methods applied to workproducts	How the completed system is validated against the original requirements
SARA	Syntactic & semantic consistency checks: (QA requirements document & module interface definition) Behavior models Interactive simulation Evaluation using the Test Environment Control flow analysis of Test Environment	Test environment (on design models and actual system)
SA-SD	Author/user cycle: (Structured spcification) Structured walkthroughs	"By comparing completed system with original structured specifications"
SD	Author/reader cycle	
SDM	Author/reader cycle Structured walkthroughs Inspections Testing Formal verification	"Acceptance tests at three or more levels are recommended"
SEPN	Structured walkthroughs Static & dynamic modeling Formal (customer) review Periodic management review Formal testing	Internal "independent test" Formal customer acceptance test
SREM	Design reviews Automatic data flow analysis of R-nets Static consistency/ completion checks on requirements database	Dynamic validation of performance requirements using simulation and post processing
STRADIS	Software QA plan matrix Structured walkthroughs: small workproducts Technical/management reviews: major deliverables Use of specialists/auditors: to review database design, test cases, requirements	Each version of system acceptance tested: Test cases (from detailed logic requirements applied to each system version Heavy user involvement in process of validation
USE	Structured walkthroughs: design Transition diagrams Consistency checks	"Not prescribed by the methodology; both verification & testing are feasible"

Table 8: Quality Assurance Methods
(Continued)

tight coupling may also contribute to difficulties in portability -- of all or part of the methodology -- thereby diminishing the desirability of the methodology.

The results of the Automated Support section of the questionnaire are presented below in two subsections: Automated Aspects and Detailed Summary of Specialized Automated Support.

Automated Aspects

Table 9 summarizes the responses to all the questions of the Automated Support section. It should be noted that no attempt has been made to standardize or clarify the table entries for "Equipment Required", which ranged from minicomputers to mainframes. Thus, someone wishing to use a specific tool set would have to make further investigation into the specific hardware/software environments in which the tools may be used.

Half of the methodologies (12 of 24) have specialized automated support, with another four planning for it in the future. It was *very* difficult to determine how "specialized" that automated support was. A positive response could denote a very limited tool set.

Most of the methodologies with tool support (10 of 12), make their tools are publicly available. However, only 2 of the tool sets are in the public domain. Half (6 of 12) of the methodologies with tool support expect problems with portability of their tools. In general, then, one should expect considerable cost and/or effort in acquiring the tools associated with a methodology.

Detailed Summary of Specialized Automated Support

Table 10 gives a detailed overview of the automated support identified for each methodology. It was difficult to determine the tool class from the name of the tool -- all that was asked was the name and an optional/required indication. Respondents rarely stated whether use of the tool was "optional" or "required".

Most of the automated tool support is for document preparation. However, it is not entirely clear that the support was "explicitly developed to support [your] the methodology". (The mention of operating systems/programming environments in Table 9 suggests that specialized documentation support was not developed; however, the responses were too sparse to determine this with certainty.)

There are also efforts in other areas, e.g., consistency checking, testing, validation, prototyping, simulation, code generation, and automatic documentation generation. However, we cannot judge their "specialized" nature (as mentioned above), nor their effectiveness, from the information obtained. Their mention is worthy of note, though, to show the "state of the art".

MANAGEMENT ASPECTS

Table 11 summarizes the questions in the Management Aspects portion of the questionnaire. While some methodologies directly address management issues, others explicitly do not. One respondent noted that "one of the prime values of the methodology is that it can be brought into an environment and not adversely impact the management philosophy".

However, a methodology that prescribes "nothing" for managing people and projects may have an adverse impact. We believe a methodology should be an integrated system of technical and managerial procedures, since the technical work is done by human beings who must be managed and who must coordinate their activities with others.

Most of the methodologies (15 of 24) claim to directly address management issues. However, in general, it appears that even when a methodology attempts to give management support, the extent of that support varies considerably. Only 9 of the 24 methodologies attempt to cover all three management issues: project, technical, and people. Of those that cover management issues at all, nearly all attempt technical management issues (13 of 15); project management issues are often coupled with them as well (12 of 13). Of those that address people management (11 of 15), most are in terms of prescribing "team" organizations (8 of 11). A small number (3 of 11) use "matrix" organization.

USAGE ASPECTS

The questions on a methodology's usage were aimed at discovering how easy the methodology has been to use, and to determine the extent of its use. Presumably, if a methodology has been used in numerous organizations, or on many projects, there has been sufficient feedback on its ease of use to

AUTOMATION ASPECTS							
Methodology	Specific Tool Support?	Equipment Required	Tools publicly available?	Portability			
				Easy	Some problems	Signif problems	Not Easy
ACM/PCM	no						
DADES	TBD						
DSSAD	TBD		no	x			
DSSD	yes	IBM, Honeywell, Univac, Perkin-Elmer	yes	x			
EDM	TBD	Dec 2060, Vax 11	not yet				
GEIS	yes	TI DS990, DX10 w/ Cobol	in 1983			x	
HOS	yes	Vax CPU with VMS O/S with >1MB memory, 100MB disc; VT-100 terminals with Digital Engineering Retro-Graph board	licensed only	x			
IBMFSD-SEP	no	S/370 MVS					x
IESM	TBD		will be		x		
ISAC	no						
JSD	TBD						
MERISE	yes	IBM370,38; CHB64,66; Univac VS9, 1100; CDC Cyber 170; Burroughs B4700, B6800, C74; Siemens BS200	yes		$	$	
NIAM	yes	CPU, S/W environment, tools, plotters, graphics terminal	yes			x	

Table 9: Automated Aspects

AUTOMATION ASPECTS							
Methodology	Specific Tool Support?	Equipment Required	Tools publicly available?	Portability			
				Easy	Some problems	Signif problems	Not Easy
PRADOS	TBD	Unix O/S	Not yet.				
REMORA	yes	Pascal compiler & standard O/S	yes	x			
SADT	yes	CDC Cybernet,DEC PDP-11, Unix V7	yes			x	
SARA	yes	MIT-Multics, Vax-Berkeley Unix (TM) (TBD)	yes*			x	
SA-SD	no**			x			
SD	no						
SDM	no						
SEPN	yes	Unix/PWB on Vax 11/780; PSL/PSA; AMDAHL for simu-lators, compilers	no		x		
SREM	yes	CDC 7600, Cyber 74/174/175; Vax 11/780; graphics consoles, Pascal/Fortran compilers; plotters	yes*			x	
STRADIS	yes	Tektronix 4113 display unit; Tek-tronix 4653 hard copy unit; 256KB extra memory; TSO/MVS O/S; IBM370 compati-ble mainframe					
USE	yes	Unix V7 or 4.1 BSD	Some now; more 6/83	x			

Table 9: Automated Aspects
(Continued)

Key:

* = public domain
TBD = to be developed
$ = "depending on the environment"
** = developed by clients

DETAILED SUMMARY OF SPECIALIZED AUTOMATED SUPPORT							
Methodology	Doc. Prep.	Transfor-mation	Code Gen	Testing/ Checking	Simulation/ Prototype	DB/ Lib	Cnfg Mgmt
ACM/PCM							
DADES				TBD			
DSSAD	Design (TBD)						
DSSD	Reqmts, Specs, Design					TBD	
EDM	Design		TBD	some testing of design solution			
GEIS	Reqmts(o), Specs(o), Design(o)?					r	
HOS	Specs, Design	S->D D->C	x	Validation at each phase			
IBMFSD-SEP	Specs?, Design?			Testing			x
IESM				Testing			
ISAC							
JSD?							
MERISE			o	Bench-marks, Testing	x	o	
NIAM	Reqmts?, Specs?, Design?	R->S?? S->D?? D->C(TBD)	for DBMS		prototype	x	
PRADOS							
REMORA	Specs(o), Design(o)		x	some consist. checking; verification of syntax & semantics	simulation (o)	x	x

Table 10: Detailed Summary of Specialized Automated Support

DETAILED SUMMARY OF SPECIALIZED AUTOMATED SUPPORT							
Methodology	Doc. Prep.	Transfor-mation	Code Gen	Testing/ Checking	Simulation/ Prototype	DB/ Lib	Cnfg Mgmt
SADT	Reqmts?, Specs?, Design?						
SARA	Reqmts?, Specs?, Design?	R->S?? S->D?? D->C??		Test environ-ment; consist. checks	simulation (o*)		
SA-SD							
SD							
SDM							
SEPN	Reqmts?, Specs?, Design?				Static & dynamic modelling (r)	o	o
SREM	Reqmts(o/r), Specs(o), Design(o)			Data flow analysis; static consist. checks; data base checks	simulation o		
STRADIS	Design?						
USE		R->S(TBD) S->D(TBD) D->C(TBD)	x	Consist. checks	o	o	TBD

Table 10: Detailed Summary of Specialized Automated Support
(Continued)

Key:

blank = no automated support reported
TBD = to be developed
r = required tool
o = optional tool
x = tool exists (don't know if required or optional)
* = strongly recommended
? = "specialized" support questioned by reviewers
?? = was difficult to determine the nature of the transformations
R->S = Requirements -> Specifications
S->D = Specifications -> Design
D->C = Design -> Code

MANAGEMENT ASPECTS								
Methodology	Address Mgmt Issues	Range of Management Issues			Directly Address			
		Project	Tech	People	Version Control	Config Mgmt	Validate Wrkpdcts	System Evolution
ACM/PCM	no							
DADES	no							
DSSAD	no							
DSSD	yes	x	x	...	x		x	yes
EDM	yes	x	x	team			x	
GEIS	no							
HOS	yes	x	x				x	yes
IBMFSD-SEP	yes	x	x	x	x	x	x	CR
IESM	yes	x	x		x		x	yes
ISAC	yes	x	x	team			x	
JSD	no							
MERISE	yes	x	x	team, matrix	x	x		x
NIAM	yes		x	matrix	x		x	yes
PRADOS	yes	x	x	team	x			
REMORA	no							
SADT	yes	x	x	team			x	
SARA	no							
SA-SD	yes		x				x	
SD	no							
SDM	yes	x		team		x	x	yes
SEPN	yes	x	x	team	x	x	x	CR
SREM	yes	x	x	team		x		yes
STRADIS	yes	x	x	matrix	x	x	x	yes
USE	yes	...	...		x	x		yes

Table 11: Management Aspects

Key:

... = no answer
blank = not provided
CR = customer responsibility
x = supports management aspect

give some confidence to the rating provided by the respondent.

The responses to this section are summarized in Table 12. The preparers, in general, judge their methodologies to be easy or moderately easy to use. Only one respondent judged the use as difficult, noting that according to our frame of reference, *any* methodology would be difficult.

Most of the methodologies have limited usage experience when measured by numbers of projects and diffusion into other organizations. Nearly half of the methodologies (11 of 24) have been used on 10 or fewer projects. Of these, two have been used only in academic environments on student projects. From other information in the questionnaires, we deduced that at least 7 of 24 are being developed in academic environments. They are identified in the table by "$". The projects they report could also have been student projects; our survey did not obtain that information, however. It might have been useful to discriminate experience on "student" and "industry" projects.

Only about one-third (9 of 24) of the methodologies have been used in more than ten (>10) organizations. All but one of those methodologies were developed by non-academic organizations.

It is unclear from the answers how many organizations have used a methodology on more than one project. Such replies would have been useful, if we had anticipated it, since repeated use of a methodology signifies some degree of satisfaction with it.

TRANSFERABILITY

Factors having a significant impact on how easily a methodology is "diffused" into organizations can be classified under the term "Transferability". These factors include the public availability of the methodology itself, of reading material describing it, the availability of seminars or consultancy assistance, and the average time that it takes for people and organizations to assimilate its ideas and methods.

Table 13 summarizes the questions in the Transferability section. There was a little overlap between the questions on seminars, so the representation is a collapsed set.

Most of the methodologies surveyed are publicly available, with only 4 of the 24 not accessible to the public. Two-thirds (16 of 24) are the subjects of reading materials that are easily obtained. Moreover, consultancy and private seminar offerings are widespread (19 of 24, in both cases). Public seminar offerings exist for more than four-tenths (10 of 24). Finally, licensing arrangements for documentation can be made in one-third (9 of 24) of the cases.

Of the 24 respondents, eleven (11) estimate that developer training time would be 1-3 months; and, another eleven (11) estimate that developers need at most 1 month of training. Only 2 other respondents anticipated more than 3 months' training time. The expected training time for managers, for all methodologies except NIAM, was the same or less as for developers. On the other hand, the times given for an organization to learn a methodology were, on the average, two to three times greater than for the developers. One-third (8 of 24) expect it to take more than 6 months for an organization to "learn the methodology".

4. CRITIQUE OF THE SURVEY

As we reflect on the survey and the responses, we see elements that proved to be very useful, along with others that we would change should we prepare another similar questionnaire. The most obvious criticisms focus on the format of the questions.

The objective questions, while designed to be easier to report, did not provide as much useful information as we had hoped. We often felt that our terms were misinterpreted or that the developer of the methodology did not relate accurately to a question. This effect was particularly noticeable in the responses to "ease of use."

The free-form responses, on the other hand, allowed the preparer to say as little or as much as was desired. In general, responses were very rich in detail. However, the loosely structured answers made the job of reporting the results more difficult. To complicate things even more, our occasional parenthetical remarks or examples, to give a frame of reference, usually caused the question to be answered only in those terms. Moreover, some questions were answered with attachments of existing documentation and, in one case, sales literature. This material then had to be sifted for relevant information. Conversely, when a free-form question was answered more briefly, there was a tendency to lapse into the jargon of the methodology, falling short of providing clear data. Thus, responses to the

USAGE ASPECTS BY METHODOLOGY						
Methodology	Degree of Difficulty				Usage Experience	
	Easy to Use	Moderate Ease	Moderate Difficulty	Difficult to Use	# of Projects	# of Organiz'tns
ACM/PCM$			x		2-10	none
DADES$			x		2-10*	none
DSSAD		x			2-10	2-10
DSSD		x			>10	>10
EDM$		x			>10	2-10
GEIS	x				2-10	none
HOS	...	...	...	...	>10	2-10
IBMFSD-SEP		x			2-10	2-10
IESM	x				2-10	2-10
ISAC		x			>10	>10
JSD		x			2-10	2-10
MERISE		x			>10	>10
NIAM		x			>10	>10
PRADOS		x			2-10	2-10
REMORA$	x				2-10	2-10
SADT	x				>10	>10
SARA$			x		>10**	2-10
SA-SD		x			>10	>10
SD$	x	x			2-10*	none
SDM			x		>10	>10
SEPN				x	>10	>10%
SREM			x		>10	>10
STRADIS		x			>10	>10
USE$		x			2-10	2-10

Table 12: Usage Aspects by Methodology

Key:

... = no answer
* = only student projects
** = "primarily" student projects
% = 15-20 "organizations" within parent organization
$ = currently in a university development setting

TRANSFERABILITY ASPECTS									
Methodology	Is it publicly available?	Teaching Aides/Devices			Seminar Offerings		Average Months Training Time$		
		Public Docs	Licn'd Docs	Consult-ancy	Pub	Priv	Dvlpr	Mgr	Org
ACM/PCM	yes	x		x			1-3	1-3	6
DADES	yes	x					1-3	<1	3
DSSAD	yes			x	x	x	1-3	<1	6
DSSD	yes	x	x	x	x	x	1-3	<1	6-12
EDM	yes	x		x	x	x	<1	<1	...
GEIS	yes		x	x		x	<1	<1	
HOS	yes	x	x	x		x	<1	<1	varies
IBMFSD-SEP	no	x					1-3	1-3	3
IESM	yes	x	x	x	x	x	<1	<1	<1
ISAC	yes	x		x		x	1-3	1-3	1-3
JSD	yes	x	x	x	x	x	<1	...	3
MERISE	yes	x	x	x	x	x	<1	<1	6
NIAM	yes	x	x	x	x	x	1-3	3-6	12
PRADOS	yes		x	x		x	1-3	1-3	...
REMORA	yes	x		x	x	x	<1	<1	1-3
SADT	yes					x	<1	<1	1
SARA	yes	x		x	x	x	1-3	1-3	6-9
SA-SD	yes	x		x		x	1-3	1-3	1-3
SD	no						<1	<1	...
SDM	yes	x		x		x	1-3	<1	1
SEPN	no						>6	>6	12
SREM	no			x		x	<1	<1	1-2
STRADIS	yes		x	x	x	x	<1	<1	<1
USE	yes	x		x		x	1-3	1-3	6

Table 13: Transferability Aspects

Key:

blank = no provision
... = no answer

free form questions had to be treated in an impressionistic and subjective way.

Another problem was confusion about terminology. There was specific difficulty in understanding whether what *we* meant by "requirements" and "specifications" in the questionnaire was what the respondent meant. It was also unclear whether "Requirements Analysis" and "Functional Specification" were viewed as distinct phases, each with specific outputs, i.e. "requirements definition" and "functional specifications".

The terms "requirements", "functional requirements", "requirements specifications", "specification", "system requirements", were all used by the respondents to label the initial phase(s) (and output) of the life cycle. The interchangeable and combinatorial use of "requirements" and "specifications" leads us to believe that there is not yet a consensus on the discrete processes involved in the early stages of the life cycle. Similarly, it was difficult to determine whether the Design phase included Detailed Design as well as Architectural Design.

As a result, we would have done better to include some standard definitions of these terms to give a common basis for discussing these phases.

5. CONCLUSION

Several points stand out in our survey of software development methods:

(1) Some methodologies are process oriented while others are data oriented; few methodologies integrate process and data orientations in more than a cursory way;

(2) Most methodologies are quite new and have seen little widespread use; many are still at the research stage;

(3) Many issues were not well addressed by the results we obtained, especially tools, management, and training;

(4) It is clear that we have missed some methodologies through oversight, ignorance, or inability to obtain information from their developers;

(5) We believe that developers answered the questionnaire truthfully; nonetheless, it is clear that more objective information on methodologies is needed, especially in the areas of applicability and transferability;

If one steps back from the field, the range of methodologies is not really very great; this implies that an integration of several different methodologies may be quite possible (necessary?).

In summary, though, we feel that our questionnaire accurately captured the "state of the art". No one methodology stands out as being "superior", with strengths and weaknesses observed in many. As with other attempts to survey and classify methodologies, the lack of well-understood dimensions severely limits one's ability to compare and contrast them. We hope that these results will help in the evolution of methodologies and the development of new ones that satisfy the requirements for methodologies cited in "Ada Methodologies: Concepts and Requirements."

IDENTIFICATION ASPECTS (from returned questionnaires)			
Methodology	Developer	Other Contacts	
		Questionnaire Preparer	Future Queries
ACM/PCM	Michael Brodie Computer Corporation of America 575 Technology Square Cambridge, MA 02139 tel: (617) 491-3670	same	same
DADES	Antoni Olivé Facultat D'Informatica Univ. Politecnica de Barcelona c/ Jordi Girona Salgado, 31 Barcelona (34) Catalonia SPAIN tel: 34 + (93) 2048252 (x 294)	same	same
DAISEE	Arne Solvberg Dept. Computer Science Univ. of Trondheim 7034 Trondheim-NTH NORWAY tel: 47 + (7) 993438	same	same
DCDSM	J. Mack Alford TRW, Huntsville Laboratory 213 Wynn Drive Huntsville, AL 35805 tel: (205) 830-3214	same	same
DSSAD	Colin Knight 23 Carlton Road Caversham, Reading RG4 7NT ENGLAND tel: 44 + (734) 470440	same	same
DSSD	Ken Orr & Assoc., Inc. 715 East 8th Street Topeka, KS 66607 tel: (913) 233-2349 (800) 255-2459	J.Highsmith	J.Highsmith
EDM	George Rzevski School of E.E. & Comp.Sci. Kingston Polytechnic Kingston-upon-Thames ENGLAND	same	same

Appendix 1 -- Identification of Methodology Developers

IDENTIFICATION ASPECTS (from returned questionnaires)			
Methodology	Developer	Other Contacts	
		Questionnaire Preparer	Future Queries
GEIS	Veikko Keha Pursimiehenkatu 3 A 7 00150 Helsinki 15 FINLAND tel: 358 + (0) 637324	same	same
HOS	Higher Order Software, Inc. 806 Massachusetts Ave. Cambridge, MA 02139 tel: (617) 661-8900	same	Tech: R.Smaby Mgmt: R.Mace
IBMFSD-SEP	Donald O'Neill IBM Federal Systems Division 6600 Rockledge Drive Bethesda, MD 20814 tel: (301) 428-2197	same	same
IESM	Dr. Basil N. Barnett DMW Information Resource Management, Ltd. Spa House 11/17 Worple Road Wimbledon, London SW19 4JS ENGLAND tel: 44 + (1) 9469109	same	same
ISAC	Mats Lundeberg Institute for Development of Activities in Organizations Stureplan 6, 4 tr S-114 35 Stockholm SWEDEN tel: 46 + (8) 233990	same	same
JSD	Michael Jackson Michael Jackson Systems Ltd. 17 Conduit Street London W1R 9TD ENGLAND tel: 44 + (1) 4996655	same	same

Appendix 1 -- Identification of Methodology Developers
(Continued)

IDENTIFICATION ASPECTS (from returned questionnaires)			
Methodology	Developer	Other Contacts	
		Questionnaire Preparer	Future Queries
MERISE	1st release: A. Rochfeld SIS - 20 20 Place Napoleon 1 92080 Paris La Defense FRANCE tel: 33 + (1) 7764302	A. Rochfeld G. Panet (SEMA)	A. Rochfeld G. Panet SEMA - 16-18 rue Barbes 92126 Montrouge Cedex FRANCE tel: 33 + (1) 6571300
NIAM	International Center for IAS-Development Control Data Belgium Raketstraat 50 1130 Brussels BELGIUM tel: 32 + (2) 2421080	R. Meersman (Control Data Belgium) G. Verheyen (Control Data B.V. Holland) J.C. Van Markenlaan 5 P.O. Box 111 2280 AC Rijswijk HOLLAND tel: 31 + (70) 949344	G.J.A. Liesveld Control Data Belgium
PRADOS	SCS Scientific Control Systems G.m.b.H Oehleckerring 40 2000 Hamburg 62 WEST GERMANY tel: 49 + (40) 531030	Kurt Hellinga SCS G.m.b.H III. Hagen 43 4300 Essen 1 WEST GERMANY tel: 49 + (201) 233091	same
REMORA	Collette Rolland Univ. Paris Sorbonne 12 Place du Pantheon 75231 Paris Cedex 05 FRANCE tel: 33 + (1) 3292140 (x612)	same	same
SADT	SofTech, Inc. 460 Totten Pond Rd. Waltham, MA 02154 tel: (617) 890-6900	Clarence G. Feldmann Douglas T. Ross	same
SARA	Prof. Gerald Estrin Computer Science Dept. 3732 Boelter Hall, UCLA Los Angeles, CA 90024 tel: (213) 825-2786	Ms. Mary Vernon Dr. Gerald Estrin	M. Vernon D. Berry B. Bussell G. Estrin

Appendix 1 -- Identification of Methodology Developers
(Continued)

IDENTIFICATION ASPECTS (from returned questionnaires)			
Methodology	Developer	Other Contacts	
		Questionnaire Preparer	Future Queries
SA-SD	T. DeMarco, E. Yourdon, L. Constantine, Yourdon, Inc. 1133 Avenue of the Americas New York, New York 10036 tel: (212) 730-2670	David M. Bulman Pragmatics, Inc. 3032 Masters Place San Diego, CA 92123 tel: (714) 565-0565	D. Bulman
SD	Hannu Kangassalo Univ. of Tampere Dept. of Math. Sciences/ Computer Science P.O. Box 607 33101 Tampere 10 FINLAND tel: 358 + (31) 156778 358 + (31) 681636	same	same
SDM	Gerald F. Hice w/ Dr. W.S. Turner L.F. Cashwell CAP Gemini, Inc. 301 Maple Ave. W. Tower Building Vienna, VA 22115 tel: (703) 938-2207	same	same
SEPN	Software Engineering Div. Hughes Aircraft Company Box 3310 Fullerton, CA 92634	T. Snyder, SEPN Director tel: (714) 732-2922 R.R. Willis tel: (714) 732-1488	R.R. Willis
SREM	J. Mack Alford TRW, Huntsville Laboratory 213 Wynn Dr. Huntsville, AL 35808 tel: (205) 837-2400	same	same
SSDM	Colin J. Tully Dept. of Computer Science University of York Heslington York YO1 5DD ENGLAND tel: 44 + (904) 59861	same	same

Appendix 1 -- Identification of Methodology Developers
(Continued)

IDENTIFICATION ASPECTS (from returned questionnaires)			
Methodology	Developer	Other Contacts	
		Questionnaire Preparer	Future Queries
STRADIS	McDonnell Automation Co. (acquired from Improved System Technologies) Dept. K277 Building 302, Level 1E P.O.Box 516 St. Louis, MO 63166 tel: (314) 233-2626	James E. Armstrong	same
USE	Anthony I. Wasserman Medical Information Science Room A-16 Univ. of California, San Francisco San Francisco, CA 94143 tel: (415) 666-2951	same	same

Appendix 1 -- Identification of Methodology Developers
(Continued)

Appendix 2: Original Questionnaire & Cover Letter

UNIVERSITY OF CALIFORNIA, SAN FRANCISCO

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

SECTION ON
MEDICAL INFORMATION SCIENCE
(415) 666-2951

SAN FRANCISCO, CALIFORNIA 94143

15 June 1982

Dear Colleague:

We are presently conducting a study of software development methodologies for the Ada Joint Program Office (AJPO) (U.S. Department of Defense). AJPO has asked us to perform this study to determine the present state of methodologies and to identify desirable characteristics of such methodologies, giving attention to compatibility with Ada and Ada Programming Support Environments.

One key step of that study is to conduct a survey of existing methodologies. We have prepared the enclosed questionnaire toward that end, and hope that you will be able to assist us by completing and returning the questionnaire at your earliest opportunity (no later than 1 August).

We recognize that it is a lengthy questionnaire and that not all aspects of it will be applicable for all of the different methodologies that we are trying to survey. Accordingly, we have divided the questionnaire into sections, and hope that you will complete as many of these sections as are relevant to your work.

We are particularly interested in obtaining the general information on the methodology, along with methods of transferability. If you can send us (one copy to each, if possible) a complete set of documentation on your methodology, our study will be greatly facilitated.

We have also included a Preface to the questionnaire that describes our outlook on methodologies, and hope that this Preface will serve to motivate, as well as to explain why we have chosen this particular organization.

We will greatly appreciate any help that you can give us, and will send you results on this survey during the Fall. Please feel free to contact either one of us using the mail or electronic mail addresses shown below if you have any questions or comments.

Thank you very much for your help.

Very truly yours,

Anthony I. Wasserman

Peter Freeman

Prof. Anthony I. Wasserman
Medical Information Science
University of California
San Francisco, CA 94143

(415) 666-2951

ARPAnet: waserman@berkeley
UUCP: ucbvax!waserman

Prof. Peter Freeman
Information and Computer Science
University of California
Irvine, CA 92717

(714) 833-6064

freeman@usc-eclb
ucbvax!ucivax!freeman

SOFTWARE DEVELOPMENT METHODOLOGY

Preface

The purpose of this questionnaire is to obtain basic information on existing software development methodologies, particularly as they may be used in conjunction with programs to be written in Ada.

Our view is that a methodology must support a process covering the entire software development activity, from the original concept of the problem through the release of a properly functioning system, and should assist with the ongoing evolution of the system. A methodology thus must include both technical methods and management procedures that are employed within a development environment. The methodology may also be supported by automated tools.

Application of the methodology yields one or more work products, such as specifications, design documents, and program code. These work products are used to assure the quality of the system and to support future evolution of the system.

We recognize that approaches to software development vary widely, and that creators of methodologies have focused on different aspects. We are aware that some approaches emphasize certain parts of the overall software development process, while omitting others. Similarly, we have seen that approaches vary in their relative treatment of technical issues, management, and organizational topics.

Accordingly, the questionnaire gives primary attention to general issues rather than seeking low level details concerning the application of the methodology. We have attempted to keep the size of the questionnaire to a minimum.

If you need more information, or have any questions, please contact either:

Anthony I. Wasserman
Medical Information Science
University of California, San Francisco
San Francisco, CA 94143

(415) 666-2951
ARPAnet: waserman@berkeley

Peter Freeman
Information and Computer Sciences
University of California, Irvine
Irvine, CA 92717

(714) 833-7403
freeman@usc-eclb

SOFTWARE DEVELOPMENT METHODOLOGY

QUESTIONNAIRE

IDENTIFICATION

Name of methodology:

Name, address, organization, and telephone number of developer:

Name, address, organization, and telephone number of person completing this questionnaire (if different from above):

Who should be the primary contact for further information?

GENERAL METHODOLOGY ISSUES

1) What are the key concepts or underlying principles that guided the creation of the methodology or its application?

2) What phases of the software development process are covered by your methodology (e.g., specification, design, verification, coding, prototype construction)?

3) To what extent does your methodology form a *system* for developing systems?

___ does so in the context of one phase (e.g. design)
___ does so in the context of several phases
___ does so if coupled with other methodologies (list below)

___ issue not addressed

4) For each of the following application areas, please indicate the relative suitability of your methodology using (1) well suited, (2) satisfactory, (3) inappropriate:

___ embedded systems/process control/device control
___ scientific/engineering systems
___ systems programming
___ software tools
___ data processing/database systems
___ expert systems/artificial intelligence

5) Is the methodology appropriate for

___ small systems (< 6 person-months; < 2000 lines of Ada code)
___ medium systems (up to 3 person-years; 2000 - 10,000 lines)
___ larger systems

1) METHODOLOGY OVERVIEW
What methods are spelled out to guide the development process? (If possible, identify a sequence of steps, including iteration where appropriate, that one should follow).

2) TECHNICAL CONCEPTS
For each of the following concepts, how (if at all) does your methodology support the concept?

a) function hierarchy/decomposition

b) data hierarchy/data abstraction

c) interface definitions

d) data flow

e) sequential control flow

f) concurrency/parallelism

g) formal program verification

3) WORK PRODUCTS

a) What work products are prescribed for users of the methodology, and how are they defined?

b) What representation schemes (e.g., axioms, data flow diagrams, finite state diagrams) are used?

c) What additional forms of documentation are recommended? Is there a prescribed format for documentation? (Please attach copies of forms or recommended formats.)

4) ADA COMPATIBILITY

Assume that your methodology is being used for the specification and design of a system that will be programmed in Ada. For each of the following Ada features, how (if at all) does your methodology facilitate mapping your specification or design into Ada?

a) packages (data abstraction)

b) tasks

c) generics (packages, types)

d) exception-handling

e) machine representation

Are there any important aspects of your specification or design methods that do *not* map well to Ada? If so, name them.

5) QUALITY ASSURANCE

For each work product, state the method, if any, used to assure the quality of the product (examples: author/reader cycle, structured walkthroughs, inspections, testing, formal verification).

What is the explicit means by which the completed system is validated against the original requirements?

AUTOMATED SUPPORT

1) Please list the names of any automated tools/environments that have been explicitly developed to support your methodology. For each tool, indicate whether its use is optional or mandatory.

2) What equipment and/or facilities (hardware, operating system, computer, etc.) are needed to use these tools?

3) Are the tools publicly available? Briefly describe acquisition method and costs.

4) Please estimate the portability of your tools.
   ___ easy to transfer to other environments
   ___ some problems (< 1 month work)
   ___ significant problems (1-12 months work)
   ___ not easily portable

MANAGEMENT ASPECTS

1) Does the methodology *specifically* address management issues?
Yes ___ No ___
(If no, skip the remainder of this section.)

2) Indicate the range of management issues:
___ project management only (examples: budget, schedule)
___ technical management only (examples: quality of design, completeness of specs)
___ comprehensive (both)

3) What are the basic approaches used for managing people? (Examples: team organizations, matrix management.)

4) What are the basic approaches used for managing the product? (Examples: validated workproducts, configuration management, version control)

5) Does the methodology specifically address software configuration management? If so, how?

6) Does it specifically address maintenance (evolution) management? If so, how?

USAGE ASPECTS

1) How would you characterize the usability of the methodology?

___ easy to use
___ moderately easy
___ moderately difficult
___ difficult

(Relative difficulty may be characterized by the number of formal steps that must be followed and by the level of specialized education/training needed by the developer.)

2) How many projects have relied primarily upon the methodology?

___ none ___ 1 ___ 2-10 ___ more than 10

3) How many organizations, other than yours, have employed the methodology?

___ none ___ 1 ___ 2-10 ___ more than 10

TRANSFERABILITY

Assume that a software development organization wishes to learn and use your methodology.

1) Is the methodology publicly available? ___ yes ___ no

2) By which approaches may one learn the methodology? (Check all that apply.)

___ public documentation, e.g. textbooks, reference manuals, journal articles
___ proprietary documentation (available through licensing)
___ consultancy
___ private (in-house) seminar or workshop
___ public seminar or workshop

3) Please list the primary, publicly available sources of documentation on the methodology. Please include the publisher's name for books.

4) Are there seminars, workshops or tutorials on the methodology?

___ Regularly scheduled public sessions
___ In-house seminar by arrangement
___ Neither

5) How much training is required for the typical developer to learn the methodology?

___ less than 1 month ___ 1-3 months ___ 3-6 months ___ > 6 months

6) How much training is required for the typical project manager to learn the methodology?

___ less than 1 month ___ 1-3 months ___ 3-6 months ___ > 6 months

7) How long would it take before an organization could effectively use the methodology on a medium-sized (2-3 person year) project?

Comparing Software Design Methods for Ada: A Study Plan

Comparing Software Design Methods for Ada: a Study Plan

Peter Freeman

Information and Computer Science

University of California, Irvine

Irvine, CA 92717

Anthony I. Wasserman

Medical Information Science

University of California, San Francisco

San Francisco, CA 94143

November 1982

TABLE OF CONTENTS

MOTIVATION AND PROBLEM FORMULATION	2
DESCRIPTIVE OVERVIEW	3
STUDY PLAN	6
Prime Contractor	6
Advisory Board	6
Design Teams	8
Implementation Teams	8
IV&V Team	8
Maintenance Teams	9
Project Phases	9
Phase 1 - Initiation	9
Phase 2 - Design	10
Phase 3 - Construction	11
Phase 4 - Change	12
Phase 5 - Evaluation	12
Phase 6 - Reporting	13
DISCUSSION	13
Overall Structure	13
Prime Contractor	14
Advisory Board	14
Design Teams	14
IV&V Team	14
Maintenance Teams	14
Phase 1 - Initiation	15
Phase 2 - Design	15
Phase 3 - Construction	16
Phase 4 - Change	16
Phase 5 - Evaluation	16
Phase 6 - Reporting	17
FURTHER STUDIES	17
ACKNOWLEDGMENTS	17
APPENDIX 1 - Suggested Methods for Study	18
APPENDIX 2 - Design Problem	19
APPENDIX 3 - Observational and Data Collection Requirements	22
APPENDIX 4 - Maintenance Manual Outline	24
APPENDIX 5 - SADT Model of Project Plan	25

MOTIVATION AND PROBLEM FORMULATION

Many recommendations have been made for improving the process of software development over the past 30 years. High level languages, timesharing systems, and programming techniques are among the ideas that have been successfully used. Many other ideas have been proposed, but not used extensively, making it difficult to determine their effectiveness.

The discipline of software engineering has emerged over the past decade as a focal point for efforts to improve both the quality of software products and the process by which they are created and maintained. After initial attempts in which isolated methods were developed to address different phases of the software lifecycle, recent work has concentrated on integration of methods.

The integration of technical methods with management procedures across the software lifecycle yields a *methodology* for software development - a process that can be systematically followed from the initial system concept through product release and operation. The methodology may be supported by automated tools, which may also be integrated into a programming support environment. Finally, the methodology and the programming support environment are normally used in a work setting, one or more physical locations in which the software development occurs.

While one would like to evaluate software development methodologies objectively, it is very difficult to do so since they are dependent upon the programming support environment, the software development organization applying the methodology, and the physical setting where the organization works. As a result, there are many variables to control, including the methodology itself, the level of training of the developers in the methodology, the execution speed of software tools in the support environment, the typing speed of the developers, and the degree to which the workspace permits uninterrupted concentration on the task at hand.

Thus, it is not possible to construct precise experiments involving methodologies in the same manner as one might perform experiments in psychology, using control groups and a limited number of independent variables. Nor is it possible to conduct experiments similar to those in the natural sciences, since it is difficult to obtain meaningful, statistically significant numerical data that characterizes well what can be observed in this type of situation. However, we believe it is possible to obtain useful information from carefully designed investigations of specific aspects of methodologies.

Accordingly, we have chosen to focus on one of the most critical components of any methodology: its technical design methods. We believe that this focus is appropriate and important because of the technical imporatnce of good design, the fact that consideration of design methods is the natural next step after consideration of the programming language used for construction, and the bridge it provides to management aspects of methodologies. In this context, we have formulated a study that provides information on the following question:

> What is the effect of various software design methods on the maintainability of systems, specifically, ones that are constructed in Ada?

In short, we want to know which method helps to produce the most maintainable systems.

Furthermore, we wish to focus still further on design methods for architectural (or structural) aspects of software design. We believe that ample attention has been paid to the detailed design aspects of software development (relative to efforts expended in other areas) and believe that it is extremely important to unde stand which methods help most in establishing good software structure. The importance of structure (at the module organization level) is well understood by those who have looked deeply at the problem of software maintenance [e.g., L.A. Belady and M.M. Lehman, "A Model of Large Program Development," *IBM Systems Journal* 15,3 (1976), pp. 225-252]. What is less well understood is the effectiveness of different methods (or classes of methods) in helping software developers achieve a "good" structuring of a system (measured in terms of how easy it is to maintain the resulting system). Thus, a refined version of the key question of this study is:

> How well do various software design methods help structure systems built in Ada, as measured by the ease of maintenance of the resulting system?

There are, of course, many other aspects of software development methodologies that could be addressed, just as there are other aspects of software design methods to investigate (for example, the correctness of the resulting systems). Our choices are motivated by our perceptions of the relative importance of the various aspects at this stage of knowledge about software development and in concert with the overall objectives of the Ada program. Likewise, in the hope of making a clear advance in our understanding of methods, we have chosen to focus on software development rather than the larger sphere of system development. If this and other investigations shed light on the software development problem, then we can consider extension of those results to the systems domain.

In line with these goals, we have organized this study to:

(1) control the influence of automated support environments, software development organizations and physical workspace on the results;

(2) control differences in ability and training of developers;

(3) separate the structuring aspects of a method from the other factors of development;

(4) provide objective data on the impact of the methods;

(5) seek out ideas from various methods that might be cross-pollinated.

DESCRIPTIVE OVERVIEW

Figure 1 illustrates the overall structure and flow of information of the comparative study. This section contains an overview of the study, which is expanded in the next section.

The study concentrates on one primary issue: *the impact of alternative technical design methods on the maintainability of Ada code.* To the extent possible, all parameters of the study have been chosen to maximize the collection of objective information on this issue. The basic elements of the study include:

- experts in each of several methods (see Appendix 1) create architectural designs for a specific problem (see Appendix 2);
- each design is implemented in Ada and checked out by each of several implementation teams, resulting in multiple implementations of each design;
- each implementation is modified by each of several maintenance teams;
- the impact of the architectural design methods on the maintainability of the resulting Ada-coded systems is evaluated and reported.

Throughout the plan we have made assumptions and set parameters to control the variability of the investigation. To the extent that we have succeeded in this, the evaluation results at the end will provide new insight into the relative ability of various technical design methods to reduce the cost of system evolution.

The investigation will be managed by an organization experienced in project management and DoD procedures. This prime contractor will run the investigation "development" as though it were a normal contract; greater than normal oversight will be needed, however, to perform data collection. A separate contractor will perform an informal independent verification and validation (IV&V) function to insure that each "deliverable" (design, code, documentation, etc.) me ts the substantive requirements of the project.

A single problem (presented in Appendix 2), representative of Ada applications, has been chosen as the basis for the design and experimental study. We are not concerned here with how easy it is to learn or use different techniques, so established experts will be used to produce the best possible designs for each method. Wherever feasible, the creatorts the substantive requiremc:its of the project.

A single design problem (outlined in Appendix 2) has been chosen that is representative of a fairly broad sample of the problems for which Ada is intended. We are not concerned here with how easy it is to learn or use different techniques, so established experts will be used to produce designs that are as near perfect an application of each method as possible; wherever feasible, the creator of a

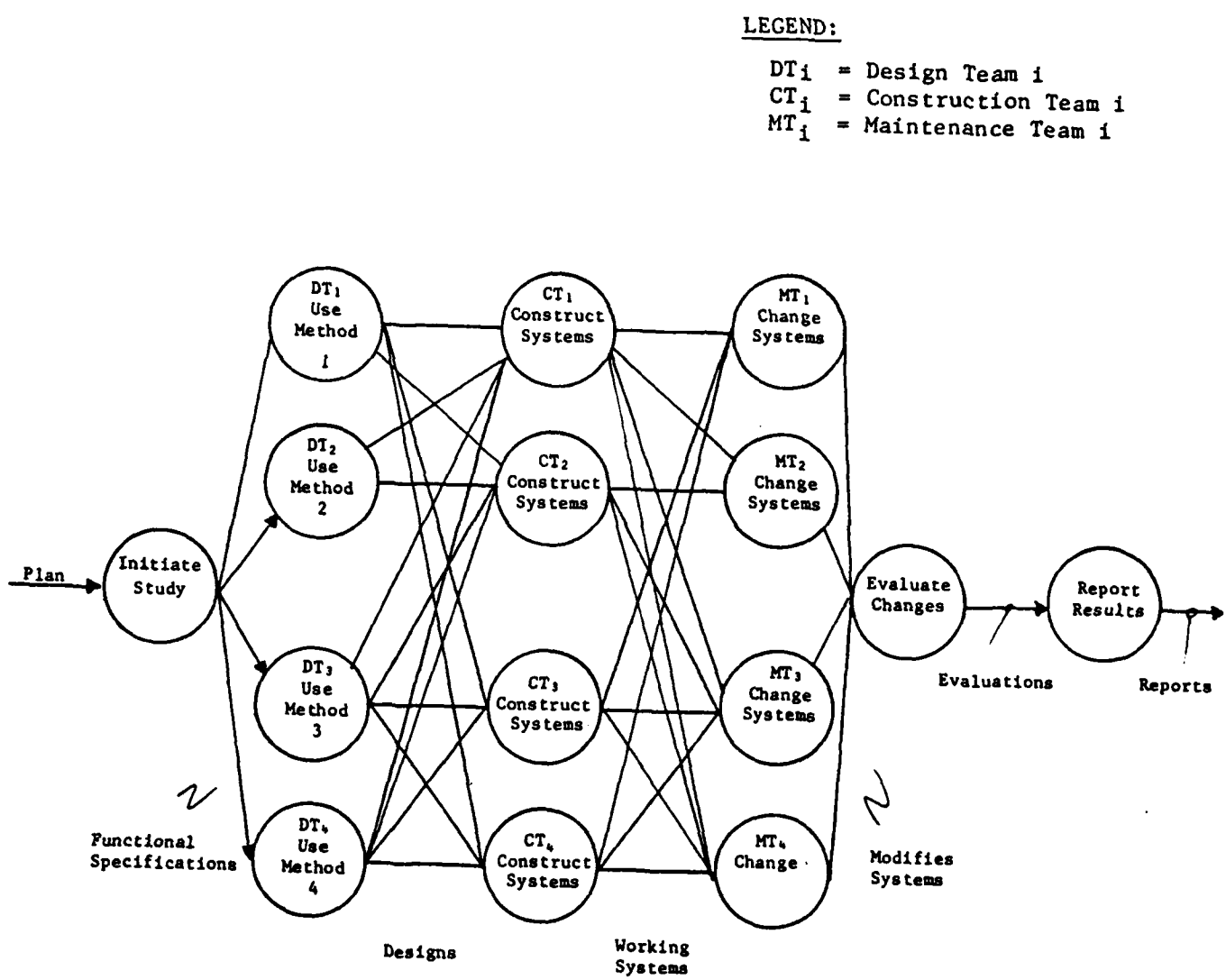


Figure 1: Comparative Study Information Flow

method will be utilized to produce the design.

Variability in implementation (caused by learning effect of multiple implemen tations by a single team and difference in ability) will be controlled by using multiple implementation teams, already expert in Ada, to code each design. To further control variability, each implementation team will implement the systems in a different order, providing a basis for statistical analysis of the resulting data. The implementation teams will be responsible for performing detailed design, coding in Ada, unit and integration testing, and validation (with the assistance of the IV&V teams) of the correctness of the resulting systems.

The starting place for the implementation efforts will be an architectural design that essentially includes the information described in section 3 of Appendix 4. The key aspects of that set of information is a specification of the modules of a system (via descriptions of their inputs, outputs, technical constraints, and brief statement of intended function) and the interconnections between them.

It is expected that implementation will uncover problems with the architectural design. Procedures will be established to record minor fixes made by the implementers, to decide on problems that should be fixed by the designers, and to permit them to make such changes.

Evolution of each system will be carried out using a maintenance manual (see Appendix 4) composed of the design and implementation documentation, and specific maintenance instructions. Multiple maintenance teams will be asked to pereful measurement of the effort required to make modifications. Secondary information will be obtained from the subjective evaluations of all parties involved in the study. (See Appendix 5)

This structural overview of the study defines the general outlines of the investigation. The next section provides detailed explanation of each aspect, a detailed project plan, and some justifications for the particular choice of investigation parameters we have made. More extensive discussion of the investigation rationale is provided in the Discussion Section.

STUDY PLAN

This section presents the proposed comparative study plan in detail; the following section, DISCUSSION, presents rationale not covered here. The overall project is broken into six phases of activity. Figure 2 indicates these major phases and shows the major project personnel involved in each phase. Our presentation is organized by first discussing each major participant and then describing each phase of activity.

Prime Contractor

The entire comparative study must be under the management control of a single organization; the substantive control of the project, however, will be shared with the Advisory Board. Specific characteristics and responsibilities include:

The contractor must be familiar with DoD procedures and (in general terms) the software development methods and philosophies involved in the study.

The contractor must not have any proprietary interest in any of the design methods in the study and cannot supply any of the other personnel (e.g. IV&V team).

The contractor should provide support and administrative direction to the Advisory Board.

The contractor must make this general plan specific by setting milestones, delivery dates, etc.

The contractor must insure that all requested data is collected.

The contractor must monitor and be responsible for the successful completion of the study.

The contractor must draft the final technical summary of the study.

The contractor publishes and disseminates results.

The contractor provides liaison with other AJPO and Ada-community activities.

Precise details of subcontracting are not our concern except that we must note that: a) the Prime Contractor must have the authority as well as the responsibility to make sure that the study is successfully completed, and; b) contracting details must not be allowed to interrupt or interfere with the substantive work in any way that would prejudice the results.

Advisory Board

The overall control of the study will be the primary responsibility of the Prime Contractor. A high-level Advisory Board will share responsibility for substantive technical decisions. We suggest a board of fewer than 10 persons composed of one or more members from:

- AJPO
- prime contractor (technical person not under the control of the project manager);
- defense contractor (not otherwise involved in the study and with no financial interest in any of the design methods);
- non-defense system developer;
- academia;
- the Ada community at large (chosen in as representative a fashion as possible).

All members should be technically competent and capable of substantively reviewing and contributing to the work being conducted in the study.

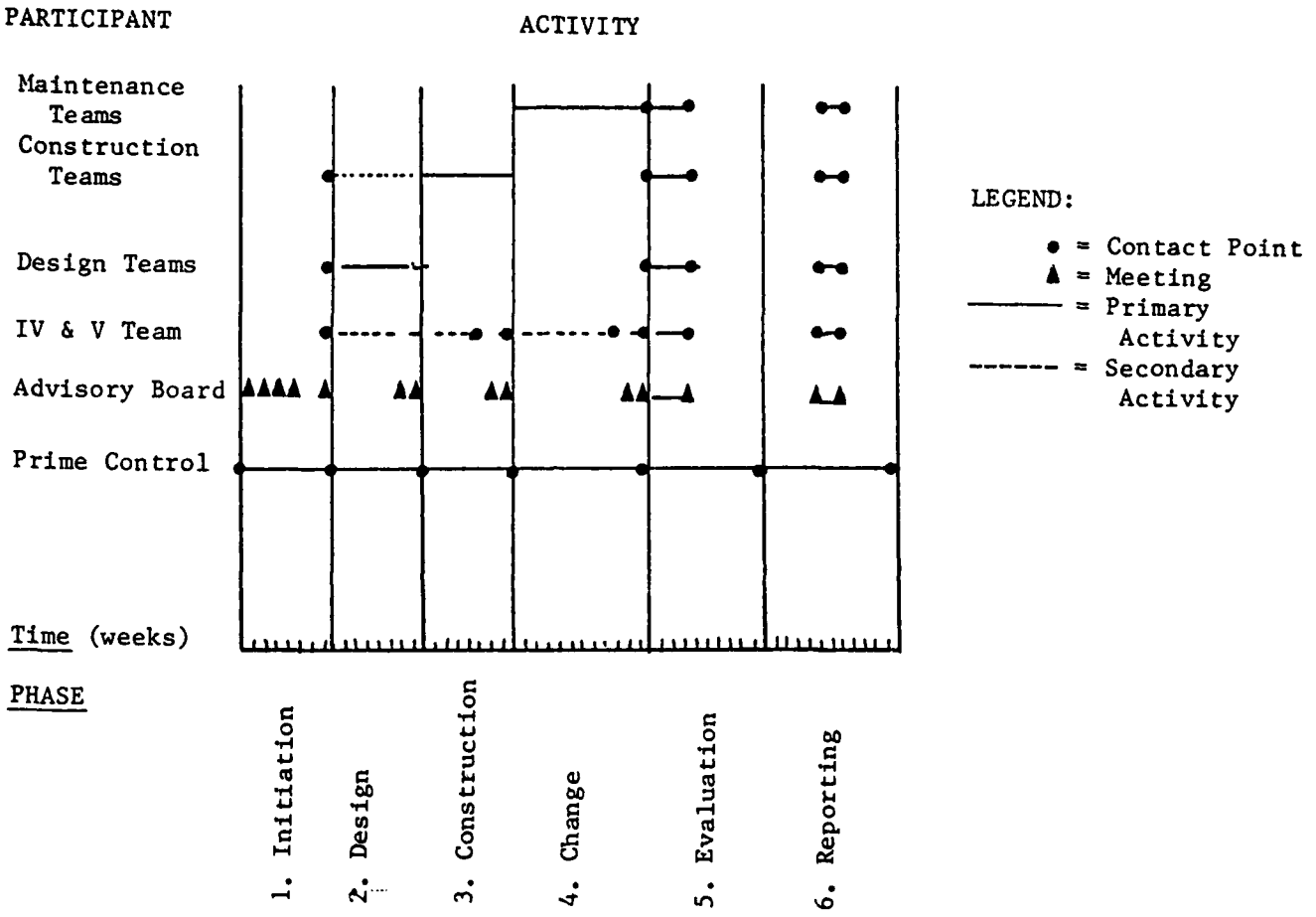


Figure 2: Study Plan

The Board should be financially and administratively supported by the Prime Contractor. A specific budget should exist for obtaining specialized assistance (e.g. someone expert in running studies of this type) if the Board decides it needs additional information in order to fulfill its review responsibilities.

The duties of the Board include:

- reviewing and approving the detailed project plan;
- reviewing and approving the choice of design methods and teams, implementation teams, maintenance teams, and IV&V teams;
- reviewing and approving the problem statement, measurements to be taken, IV&V procedures, and all other major technical decisions;
- making an independent evaluation of the data taken and presenting a written report to the prime contractor for inclusion in the final report;
- reviewing and approving the release of the final technical summary of the study;
- writing and publishing an independent evaluation of the study's methodology (to permit improvement of future studies).

Design Teams

There will be one design team per method chosen for study. Wherever possible, the design team shall include the developer of the method. The organization (number of people, management structure) of each design team is determined by that team, subject to the requirements and constraints noted below in the description of each Phase of work. In some instances, there may be only a single designer on the design team.

No design team should employ more than one method nor participate in more than one design effort that is a part of this study. No member of a design team shall participate in any other phase of the study except as noted.

The design team is responsible for preparing the relevant sections of the maintenance manual (see Appendix 4) and transmitting that manual with the design.

The objective of this phase is to obtain the best possible design in each method under reasonable project constraints.

Implementation Teams

Implementation teams will be responsible for implementing each design in Ada on a common machine. Members of the teams should be knowledgeable in Ada. Each team is responsible for delivering a final maintenance manual for each design in addition to carrying out the coding and checkout.

Each team should be highly capable in transforming designs to a uniform style of Ada code. The implementors must make sure that each design works as specified. Care must be taken to limit design changes made by the implementors and to permit limited rework of a design by the designers.

IV&V Team

This team should be independent of the methods and the other teams. Its responsibility is to ensure that the workproducts produced meet the project standards. Although a single IV&V contractor should be used, it is likely that individual teams will be assigned to each design and implementation team.

Maintenance Teams

The maintenance teams are a critical factor in the study and must be carefully chosen. Their responsibilities and characteristics include:

- making specified changes to all systems in the study;
- updating maintenance manuals to reflect the changes;
- recording required data;
- recording subjective observations on the ease of change in each system;

Members of maintenance teams should be representative of personnel that might be found in an Ada application situation:

- they should be competent in Ada coding, but not experts;
- they should not have a detailed knowledge of any design method;
- they should have experience in reading and understanding design documentation;
- they should have significant experience in maintaining software.

The attempt is to simulate a maintenance group that may be called upon to modify Ada code in an embedded system without having participated in the design or coding process, working only from a complete maintenance manual.

Project Phases

For each phase, we will define the primary focus and list the major deliverables, activities, and considerations.

Phase 1 - Initiation

Focus : Detailed planning and initiation of study.

Deliverables : Detailed project plan, detailed technical requirements, contracts with participant teams and Advisory Board.

Activities :

- form Advisory Board;
- select and contract with all participant teams; the number of design methods to be selected should be five or fewer;
- refine statement of problem(s);
- refine measurements to be taken;
- outline final report;
- establish secondary information to be collected;
- establish IV&V procedures;
- establish study monitoring mechanisms;
- establish liaison with other AJPO and Ada-community projects.

Considerations:

The Advisory Board should be involved from the start and assist in making all technical decisions.

It is advisable to choose design methods that are representative of a *class* of methods, rather than choosing competetive methods that belong to the same class.

The design problem stated in Appendix 2 is of moderate size so that the designer can illustrate the design method clearly, and so that others can comprehend the basic concepts of the design method. Ideally, the design methods should also be applied to a design problem of significant size, e.g., several months of design work, which is more characteristic of large-scale embedded systems applications. However, time and cost considerations may make it infeasible for the design methods to be applied to a large scale problem, and the results from the medium-sized problem of Appendix 2 may have to suffice.

Every care should be exercised to review the functional specfications thoroughly before design begins to prevent changes or misunderstandings during design. We suggest a meeting of all design teams and the Advisory Board before design begins.

Phase 2 - Design

Focus : Design of system by different design teams to satisfy Problem Statement (N different methods plus a control group using no explicit method)

Deliverables : One design by each team, draft of maintenance manual IV&V design report, design measurements, design observations.

Activities :

- technical design by each design team;
- observation and data collection on design activities;
- verification that each design meets the functional specification, stops short of implementation, and is representative of that particular method;
- abstracting of design information to produce draft maintenance manual.

Considerations:

An upper bound on the time for design should be established, but each design team should have sufficient time in their judgment to produce as good a design as possible using their technique; Modifying this consideration, however, is the requirement that the design should be produced in a straightforward manner with only normal time permitted for review and rework. Specifically, complete redesign and excessive polishing is not to be permitted. The Advisory Board should determine appropriate limitations.

Complete data collection and observational requirements and techniques must be established in advance -- suggestions are made in Appendix 3.

An IV&V team member should be assigned to each design team to make independent observations and to relieve designers of the burden of data collection (insofar as possible). (This consideration may not apply for a small design problem);

Any questions or changes regarding the problem requirements that arise during the design must be handled explicitly by the project monitor; such questions should be transmitted to the Prime Contractor in writing, who should prepare a written reply, with the advice of the Advisory Board if necessary;

Each design method is assumed to provide its own definition of what information (and in what form) results from applying the method. This definition must be made explicit and reviewed by the Advisory Board to insure conformity with the information outlined in Appendix 4, Section 3. In those cases where application of the design method leads to the production of executable code, it is necessary to define an artificial product that stops short of code production.

The study is not a race between methods. Rather it is a comparison of design methods of different types;

The IV&V function is intended primarily to make sure that the study requirements have been fulfilled, not to check the technical accuracy or quality of the design;

As a control on the investigation, one system should be developed using no explicit design method. An N+1st design team should be selected for this purpose and instructed to develop the system to the point where it is ready for the implementation team. It must be insured in advance (by looking at previous work of this team) that they will not use any specific method or design representation. Their design, however, should be documented in whatever way they would normally do it. (It may be difficult to find an appropriate group, since many groups that do not use an explicit design method jump directly from the functional specification to coding.)

Any major inconsistencies in the design found by the IV&V team must be corrected by the design team before that design is turned over to the implementors;

The design deliverable must contain all information about the design that is needed by the implementors. It should be documented in a complete and easy to use manner and contain any information about the design representation that is needed for implementation.

The maintenance manual is outlined in Appendix 4. The draft produced in this phase should contain all information about the design needed by the maintainers. This information should describe the design representation (charts, etc.) so that both the design documentation and the code can be maintained. The manual should not include information that is pertinent only to the implementors.

Phase 3 - Construction

Focus: Production of working Ada code for each design.

Deliverables: Compiled code, complete maintenance manual, IV&V implementation report.

Activities:

- detailed program design;
- coding;
- data collection and observation;
- IV&V of implementation;
- completion of maintenance manual.

Considerations

Each implementation must be as true to its design as possible; the implementors must not take liberties with the design even if it seems wrong, inefficient, etc.

Standard Ada coding must be employed.

Coding style guidelines will be established and followed.

A standard test must be established so that each implementation can be checked.

The IV&V will validate the correct operation.

The maintenance manual must be completed and verified to be complete for the maintenance activity.

Phase 4 - Change

Focus: Change of each system to meet a specific set of change requests.

Deliverables: Changed systems, IV&V maintenance reports, measurements and observations.

Activities:

- code changing;
- design changing;
- observation and data collection on maintenance activity;
- IV&V of changes.

Considerations:

It is critical and essential that the design and implementation teams not know the nature of the changes during original design. We suggest that the Advisory Board develop the set of changes to be made after all designs are completed but not release them until the implementations are done.

As with the implementation, the changes are to be made and tested for accuracy against a standard set of tests.

The IV&V effort will verify the changes and validate the system operation.

All changes should be reflected in both the code and the maintenance manuals to keep the design documentation consistent with the system.

Careful measurement must be made in order to differentiate the time necessary to make the change from the time necessary to change the documentation.

Phase 5 -- Evaluation

Focus : Evaluation of the impact of different designs on maintenance from several different perspectives.

Deliverables: Individual evaluation reports, final report.

Activities:

- evaluation of the correctness of changes to the design by each design team;
- evaluation of all changes to the code by the implementation teams;
- evaluation of the change process by the maintenance teams;
- evaluation of the changes by the Advisory Board;
- collection and summary of individual reports by Prime Contractor.

Considerations:

Individuals and teams involved in the study will perform an evaluation and write a report on their perceptions of the relation between the maintainability of each system and the design method used

to produce it.

The Prime Contractor has the responsibility for going beyond these individual reports, to generalize from all of the data, observations, and individual evaluations collected in earlier phases. The Advisory Board will assist in refining this final report.

Phase 6 - Reporting

Focus: Distribution of results to the technical community.

Deliverables: Distribution of reports, successful completion of conference.

Activities:

- widespread distribution of report;
- conference to present and discuss results;

Considerations

The results of this study will not be definitive and will be open to interpretation in most instances. It is essential that an open forum be held to permit discussion of the results, questioning of the methods used, and investigation of the data collecting techniques.

DISCUSSION

In this section we provide rationale for some aspects of the study presented in the previous section and discuss several alternatives.

Overall Structure

We have tried to follow the lifecycle of development currently used by many software contractors. That does not imply that we think that it is necessarily the best approach (e.g. it does not involve the use of prototyping which is increasingly seen as a valuable development tool); rather, our objective has been to make this study as realistic as possible in terms of modelling those environments where Ada development methods will be used.

We have focused on architectural design because that explicit design methods, more than anything else, can significantly improve the maintainability of the code that is later produced. Of course, other phases and activities are important. For example, management structures and technical review procedures may be even more important than any particular design method in determining maintainability. However, we felt it was well beyond the capability of controlled investigation to compare alternative management strategies. Since design immediately precedes coding in Ada, it is the natural target for study and improvement.

The overriding principle we followed in devising the entire study was *simplicity*. We feel strongly that careful investigation of a specific question can be more valuable that a more broadly-based study providing less detailed data. In support of this position, we point to the impact of Dijkstra's use of structured programming and levels of abstraction in the development of T.H.E. operating system [E.W. Dijkstra, "The Structure of the 'THE' Multiprogrmming System," *CACM* 11,5 (1968), pp. 341-346], Baker and Mills' [F.T. Baker, "System Quality Through Structured Programming," *Proc AFIPS 1972 FJCC*, pp. 339-343] use of structured programming technology in the New York Times experiment, and Parnas' use of information hiding in the design of his KWIC system [D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *CACM* 15,12 (1972) , pp 1053-1058]. None of these cases yielded statistically significant results and the results were open to interpretation (to put it mildly). Yet they have had an extremely strong impact on the thinking of software developers and have been widely adopted. A large measure of their impact was due, we believe, to their narrow focus, so that one associates project success with the particular new items of technology being employed.

We *do not* anticipate a strong differentiation between the various design methods employed. Indeed, it would not surprise us if the final conclusion is that what has been shown is that the use of *any* explicit design method produces significant improvements in maintainability. While the "true believers" know that already, a convincing demonstration would go far toward facilitating the adoption of improved methods.

We note, also, that the conduct of this study is intended to provide a clear model of the use of good software engineering project principles. This is part of the motivation for use of IV&V, external reviewers, and other devices.

Prime Contractor

A prime contractor was specified to maintain fidelity to the way business is done in the Ada-community and to have a single point of control and responsibility for the project. We have purposely left most of the details of the study in outline form to permit the Prime Contractor to refine the plan to fit the specifics of the actual investigation. This, coupled with the responsibilities of the Prime Contractor mean that the organization fulfilling that role must be highly capable of providing technical as well as managerial leadership.

Advisory Board

There are three primary reasons for having an Advisory Board. First, it provides the necessary technical overview for the entire study, performing detailed tracking of the investigation. While the Prime Contractor could do this alone, we believe that it is better to separate, at least partially, the day-to-day contract administrative matters from the technical advising so that neither is shortchanged. Second, since this study could have a large impact, it is important to get the very best guidance possible from as wide a segment of the technical community as possible. Third, the Board can serve as a resource for guidance for the Prime Contractor.

Design Teams

It would be interesting to have multiple designs based on each method in order to obtain quasi-statistical results. This, however, introduces many more problems (capabilities of different groups, for example) than it solves. As we noted above, we are striving to obtain *some* results that can be as strongly interpreted as possible. Thus, we have chosen to skirt around issues such as whether a particular method can be applied in production environments by someone other than its inventor. (All of the methods we have suggested have previously been successfully transferred.)

We do not want the design teams to get into very low-level detailed design because we want to stress the ability of different methods to facilitate good structure.

We have specified expert programmers versed in Ada to help control the effects of differing ability further. Data on the implementation process can yield information about the relative ease of implementing various design. That is not the focus of the experiment, however, because there is relatively little effort spent on that phase anyway.

IV&V Team

We have specified an IV&V team to mirror the actual development culture that is increasingly used in DoD work. Furthermore, it provides a simple way to assure that the technical work of the study conforms to plan. Finally, it will introduce an added guarantee of objectivity and, through their observational activities, partially free the development teams from the distraction of record keeping required by the nature of the study.

Maintenance Teams

It is often argued that the best programmers should be assigned to the maintenance activity. We do not disagree with that philosophy, but have observed that it rarely happens. Instead, our model has been that maintainers are faced with a broken (or inadequate) piece of equipment. They are given an instruction manual to go with their limited knowledge of the design process that created the piece of

equipment. Their task is to fix the equipment. We have used that analogy with appropriate changes to make it fit the software situation.

Phase 1 - Initiation

We noted above that we understand that this plan will need to be refined at the start of the study to make it fit the actual situation at that time and place. We would urge, however, that changes in structure or intent be made sparingly and only after careful review.

A standard procedure in an investigation of this type is to run a pilot study (perhaps on only one method) to help refine the details of this plan. We strongly recommend that this be done in this case. Methodological experimentation in general and especially in the software area is still at a rudimentary stage so that it is impossible to specify accurately data collection procedures, establish control and coordination mechanisms, and forsee other problems in advance. With a pilot study, we feel that the refined investigation can be reasonably assured of success; without one, we feel that the danger of a fatally flawed investigation is very great.

The problem selection -- an electronic mail system comparable to the UC Berkeley mail system developed in the Computer Systems Research Group -- was governed by several considerations, including the following:

- a realistic problem of moderate size, i.e., nontrivial, but not huge;
- limited expert knowledge needed;
- a problem involving real time processing and potential concurrency;
- suitable for programming in Ada, involving Ada features such as tasks, packages, and exception handling;
- balance between process design and data design;
- a number of different possible solutions.

One activity may need explication -- outlining the final report during project initiation. We are strong believers in objective-oriented activity. By outlining the final report at the start of the project those people responsible for its content (the Prime Contractor and the Advisory Board) will be forced to think through more precisely what information they must collect during the investigation.

As noted above, there is only limited experience with measurement of parameters of interest in this type of investigation. This experience, though, shows that the measurement activities should be planned from the outset and that ample time and resources should be allocated for data collection during the investigation.

Phase 2 - Design

We have tried to strike a balance between letting designers have as much time as needed to produce a "perfect" design (necessary if we want an accurate test only of the technical capabilities of the method) and reality in which there is never enough time to design to perfection.

Making observations and collecting data in this type of investigation takes a surprising amount of time and effort. Since this would not normally be a part of a development effort (we and others would argue that it *should* be, but it usually isn't) we don't want these activities to interfere with the development any more than necessary. Our solution is the introduction of an IV&V team member as an adjunct to each design team (the same should be done during implementation and change as well) to help with the data collection. All of the teams must expect to devote some effort to data collection, however.

Change during the design effort is inevitable since we are not working from formal specifications that can only be interpreted in one way. Careful review of the problem statement and a face-to-face meeting between all developers and the project management at the beginning should serve to prevent many of the problems. For those that do arise, it is essential that everyone have the information so that design won't differ because of differences in knowledge about the problem statement.

The use of a "control group" that is not using any specific design technique (indeed, it must be guaranteed that they are not using anything that even comes very close to a specific method -- we want them to use the old "seat of the pants", scribbles on an envelope approach) is almost buried in the project description above. This is not intended to downplay its significance. Rather, we wish to emphasize that having this control group is an integral part of the study plan since comparison to the old, informal way of doing things is perhaps the most important comparison that can be made. We recognize, however, that there are many "non-methods"; the Advisory Board can help define the control situation.

The requirement that the delivered design contain all information needed by the implementors stems from two factors. First, that it is a reflection of a fundamental software engineering principle. Second, it forces the designers to record their results and not leave things in their heads to be extracted (with some pain to both parties) by the implementors. The need for some interfacing and rework is noted above, however.

The same principle applies to the maintenance manual, but is even more important in that case. Maintainers of systems rarely have the luxury of direct contact with the developers -- that is one of the factors that makes maintenance so expensive. The critical issue in this investigation is to determine the impact of various methods on the maintenance of the resulting system. The impact, we hypothesize, may come through two channels -- via improved structure in the design itself and via improved structure in the documentation.

We reason that for most (if not all) of the methods that may be studied, competent programmers should be able to understand the *results* of applying the method with only a minimum of training (to be supplied by the documentation itself). Further, we believe that this same minimal training will permit the maintenance team not only to use the design to help speed up and improve the quality of the maintenance activity but also to record their changes in altered editions of the design. An interesting point raised by one researcher (Rob Kling) is that the work style of the design teams may have more influence than the method they use. While this may be true, we do not know how to control for it and can only warn the investigators to be on the lookout for such effects.

Phase 3 - Construction

We have chosen to divide the development activity at the end of architectural design for the technical and technology transfer reasons outlined above. This means that the construction teams must carry out detailed design. We have not specified the precise methods they should follow in doing this but we recommend the use of stepwise refinement using an Ada-oriented program design language (PDL).

An alternative to the entire structure of this part of the investigation would be to have each design team implement its design as well. While this would simplify the design of the investigation, we believe it would seriously degrade the information obtained as to the effectiveness of alternative design methods.

Phase 4 - Change

We believe that more than one maintenance team, representing different organizations, should be chosen. Each maintenance team would have exactly the same task -- to prepare a set of revisions to design and code delivered for *each* of the designs.

Phase 5 - Evaluation

We are asking for evaluation by as many different people as possible since we believe that the subjective evaluations will be an important adjunct to the data collected. Indeed, if several more or less independent groups concur in their subjective evaluations, then this will strengthen (or weaken) the objective data that is collected. It will also help make the results more believable to the community.

Phase 6 - Reporting

The purpose of holding a workshop to present the results is not to argue about them, but rather to permit the community to probe how the results were obtained in as much detail as desired. This will be beneficial in exposing weaknesses that can be corrected in future investigations as well as making the results more believable (because they are open to inspection) to the community.

FURTHER STUDIES

If the software field is in its infancy (as is often said), then objective investigation, let alone experimentation in the true sense, has not even been conceived. It would be superfluous for us to outline other possibilities here (a recent issue of Software Engineering Notes [vol. 7, no. 1] contains several concrete proposals), but we would note the value of replicating this (or any) investigation several times in different circumstances.

We do want to emphasize two things, however. First, we strongly believe that objective investigations of the ways in which software is developed must be carried out to permit us to make more rational decisions regarding the way we organize and carry out system development. Second, the investigative process must be refined greatly from its present state.

ACKNOWLEDGMENTS

Our work was aided by the results of the ACM SIGSOFT Symposium on Tool and Methodology Evaluation, as published in *Software Engineering Notes*, vol. 7, no. 1 (January, 1982), pp. 6-74. We appreciate the comments given on the initial draft of this document by Ruven Brooks, Bill Curtis, Rob Kling, and Ben Shneiderman; those of Deborah Boehm-Davis and Elizabeth Kruesi were especially detailed and instrumental in preparation of the final draft. We are grateful to Susan Richter for doing the artwork and assisting with the text editing and formatting.

APPENDIX 1 - Suggested Methods for Study

We have identified upwards of 40 methodologies for software development and have sent questionnaires to the creators of these methodologies. (See "Ada Methodology Questionnaire Summary," by M. Porcella, P. Freeman, and A.I. Wasserman.) While we may have overlooked some approaches, we believe that we have identified those that have been most widely used.

In selecting methodologies to be compared, we have used four subjective criteria:

1) widespread use

2) publicly available documentation

3) use of the method within DoD

4) mapping of design or specification primitives to Ada

However, not all of the recommended methodologies satisfy all of the criteria.

We also have sought to identify methods spanning a range of formalism. On that basis, we recommend that the comparative study choose from the following methodologies:

1) STRADIS (McDonnell Douglas)

2) Yourdon

3) Jackson Design Method

4) HOS (Higher Order Software)

5) ISAC (Mats Lundeberg)

6) SADT (SofTech)

7) SARA (G. Estrin, UCLA)

Other potential candidates if broader diversity is desired, are:

1) HDM (SRI International)

2) Wellmade (Honeywell)

3) User Software Engineering (A.I. Wasserman, U.C. San Francisco)

4) Warnier/Orr (Ken Orr & Associates)

5) PSL/PSA (D. Teichroew, U. of Michigan)

APPENDIX 2 -- Design Problem

This problem is divided into two parts: an initial design problem, which should be designed and coded in Ada, and a set of requested changes that should be given to a maintenance team for redesign and code modifications.

This is a moderate sized design problem, exhibiting elements of real time processing and concurrency, and necessitating both process and data design. It is suitable for programming in Ada, and represents a problem of realistic interest to the Ada community.

The moderate size is intended to strike a balance between a trivial problem that can be solved without the aid of any design methods and the mammoth design problems that characterize many of the larger command-and-control systems.

If time permits, the Prime Contractor could construct a much larger design problem for the design teams. However, we do not believe that such an exercise would be cost effective in showing the efficacy of design methods.

The Problem Statement

We wish to design and build an electronic mail system for a distributed setting. The setting consists of one or more local networks, each of which are known by an alphanumeric name of 1 to 8 characters, e.g. berkeley. These local sites are connected by common carrier to the ARPA network.

Each local site has one or more machines. If a local network has more than one machine, each is named by a single letter (upper or lower case).

Every machine has one or more users, each of whom has a login code of 1 to 8 alphanumeric letters, e.g., druffel. Each login name is unique on a machine, but the same login name may exist on any number of machines within the entire system. Thus, an individual may have any number of login names, not necessarily the same, on any number of machines at any number of sites.

Valid addresses may therefore be described by the following syntax:

[machine ":"] loginname ["@" site]

The following constructs are thus permissible:

freeman
jones@dec
v:waserman
A:good@texas

There are, within the network, a known set of sites at any time. Furthermore, there is a known set of local machine identifications at each site.

The command

mail addressee

will then accept text to be sent to the addressee at the designated site and machine. Input is accepted until the input stream receives an EOF character (ctrl-D). An alternative form is to transmit a file to the addressee. This is accomplished with the command

mail addressee < fname

where fname is a file for which the user has read/copy permission.

If mail cannot be sent to addressee, the message is returned to the person sending the message, preceded by the text

Could not send mail to "addressee" -- text follows

It is then seen by the user as a message sent to oneself.

When a user logs into a given machine, the system will print the message

You have mail.

if mail has arrived for the user (according to that login name for that machine) since that user's previous login. (Note: if necessary for your solution, you may assume that transmission of the message occurs within a day, but is not necessarily immediate.)

The user can then access the mail by typing the command

mail

with no parameters. If there are no waiting messages (the user simply typed the mail command in the absence of the "You have mail" message), the system will reply

No mail.

Otherwise, the system will produce a list of unread messages, in the following form

N sender date lines/chars

where N is an integer, sender is the login name and site of the sender of the message, date is the time that the message was received, lines tells how many lines are in the message, and chars gives the number of characters in the message. (Note that untransmitted messages, as described above, are returned in this manner.)

Thus, the list might appear

```
1 freeman@eclb       Sep 12  13:50   25/1204
2 waserman@berkeley  Sep 12  14:25   40/1723
```

The user may then issue the following commands:

N	typing a number causes the message to be printed; N must be a valid message number or a diagnostic will be printed
dN	causes message N to be deleted
s fname	causes most recently printed message to be stored in a local file named fname
h	prints the header line for all unprocessed messages
q	quit, saving any unread messages for the next login

Thus, the following sequence of commands would cause message 1 to be printed and deleted, and message 2 to be printed and saved in file berk.12Sep before quitting.

```
1
dl
2
s berk.12Sep
q
```

Required Modifications

We wish to make the following set of modifications to the system described above. The result of this set of changes should be a revised design document and revised code, along with supporting documentation showing the affected modules and the relevant design decisions. Wherever possible, the module changes should be linked to the changed requirements.

(1) Allow user names to by 1 to 10 characters instead of 1 to 8 characters.

(2) Permit a message to be sent to more than one user at a time. This may be accomplished in two different ways:

a) mail addresseelist

where addresseelist is a sequence of (1 or more) addressees with names separated by commas, e.g.,
mail a:smith, J:williams, casey@bat

b) mail alias

where alias is a predefined string in a file of aliases that creates an equivalence between the alias and a list of addressees. Thus, the entry

```
friends = a:smith, J:williams, casey@bat
```

would cause the command

```
mail friends
```

to have the same effect as the command in part a).

(3) Add the reply command (r) to permit the user to reply directly to the most recently processed message. Thus, if one had read a message from freeman@eclb, the command

```
r
```

would accept input from the user and transmit it to freeman@eclb. Similarly, the command

```
r < fname
```

would transmit the contents of file fname to freeman@eclb.

(4) Add the list command (l) to permit the user to obtain a list of all unprocessed messages in the same format as is given in response to the original mail command.

APPENDIX 3: Observational and Data Collection Requirements

NOTE: These requirements are only in outline form at this point. They must be expanded and refined before the actual study is carried out. A good reference for additional suggestions on data collection is Part V of *Tutorial: Models and Metrics for Software Management and Engineering* by Victor R. Basili (IEEE Computer Society).

We suggest the following information, at least, be collected at each stage:

DESIGN

- team information
  - names of all participants
  - technical background of each
- environment description
  - working conditions
  - on-line aids
- technical activity
  - accurate record of actual hours worked by individual
  - breakdown into categories, including
    - background research
    - meetings
    - working group sessions
    - individual technical work
    - documentation (low creative component)
    - non-project activity (interruptions, etc.)
- *** where possible, measurements should be on 1/4 hour breakdown
- technical results
  - number of modules
  - number of distinct data structures
  - pages of documentation
  - characters of documentation

CONSTRUCTION

- team information as above
- technical activity as above, but modified to include
  - coding
  - waiting for system
  - correcting compile errors
  - interpreting design
  - seeking clarification of design
- results to include
  - number of Ada statements
  - number of declarations
  - number of characters in Ada representation

CHANGE

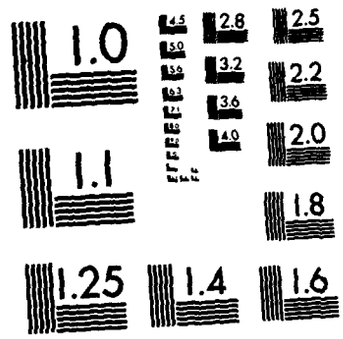
team information

technical activity
- time spent understanding change requests
- time spent understanding design
- time spent understanding code
- time spent making changes to code
- time spent making changes to design
- time spent compiling changes

technical results
- lines of code changed
- number of modules changed

AD-A123 710 SOFTWARE DEVELOPMENT METHODOLOGIES AND ADA ADA METHODOLOGIES: CONCEPTS AN..(U) CALIFORNIA UNIV IRVINE P FREEMAN ET AL. NOV 82 2/2

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX 4: Maintenance Manual Outline

The Maintenance Manual must contain all information that the maintainers of a system need in order to make changes efficiently. The precise content of the information will depend on the design method used, but the format of the document should be approximately the same for each method.

1. SYSTEM OVERVIEW

- 1.1 Basic purpose of system
- 1.2 Functional description of operation
- 1.3 Implementation considerations
- 1.4 Operational considerations
- 1.5 Guide to Maintenance Manual

2. SYSTEM SPECIFICATION

- 2.1 Complete definition of all external functions
- 2.2 Complete definition of all external data items
- 2.3 Complete definition of all internal functions
- 2.4 Complete definition of all internal data items
- 2.5 All constraints (design, implementation, operation)

3. SYSTEM STRUCTURE

- 3.1 Overview of system modularization
- 3.2 Logical data definitions
- 3.3 Module Interface definitions
- 3.4 External module definitions

4. IMPLEMENTATION

- 4.1 Detailed design of modules
- 4.2 Physical data structures
- 4.3 Code listings

APPENDIX 5 -- SADT Model of Project Plan

A complete SADT model consists of two kinds of diagrams: activity diagrams (called actigrams) and data diagrams (called diagrams). The view of an actigram is that data objects flow between activities while the view of a datagram is that activities during their operation access data objects. The only difference is the center of attention. Only actigram models will be discussed in this appendix.

THE ELEMENTS OF AN ACTIGRAM

An actigram depicts three to six activities which are represented as boxes. The limit on the number of activities depicted helps to limit the amount of information a reader of an actigram must deal with. The boxes of an actigram are connected by arrows which represent data objects. Actigrams are *data-flow* diagrams. This means that the activity of a box takes place only when the data objects represented by incoming arrows to a box are present.

The positions of the arrows on the box determines what type of data an arrow represents as shown in Figure 5.1. When input, control, and mechanism objects are present, the activity uses the mechanism as an agent to transform the input data objects into the output data objects under the guidance and constraints of the control data objects. Activity names should be verbs, while data object names should be nouns. Each activity must have at least one control and output.

A double headed dotted arrrow may be used as a shorthand in SADT to denote data relations between activities as shown in Figure 5.2.

THE STRUCTURE OF AN SADT MODEL

Each actigram is an elaboration of an activity box in a higher-level diagrsm called the parent diagrsm. If a page number appears in parentheses just outside the lower righthand corner of an activity box, then this number specifies the page of the actigram which elaborates the box. The inputs, outputs, controls, and mechanisms used in an actigram are the same as those as those on the corresponding activity box in the parent diagram. Each actigram should include from three to six activity boxes.

The highest-level actigram of a model is the only exception to the three to six activity rule and it presents only one activity, the one being modeled. The inputs, outputs, controls, and mechanisms which are used in the rest of the model are specified on this highest-level actigram called A-0. The A-0 actigram represents the context in which the system being modeled operates. As a part of the context the A-0 actigram explicitly states in prose the purpose of the model and from what viewpoint the model was made.

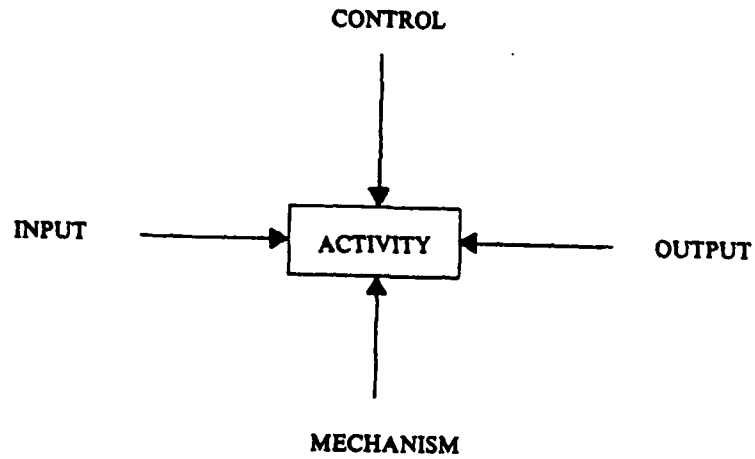


Figure 5.1.

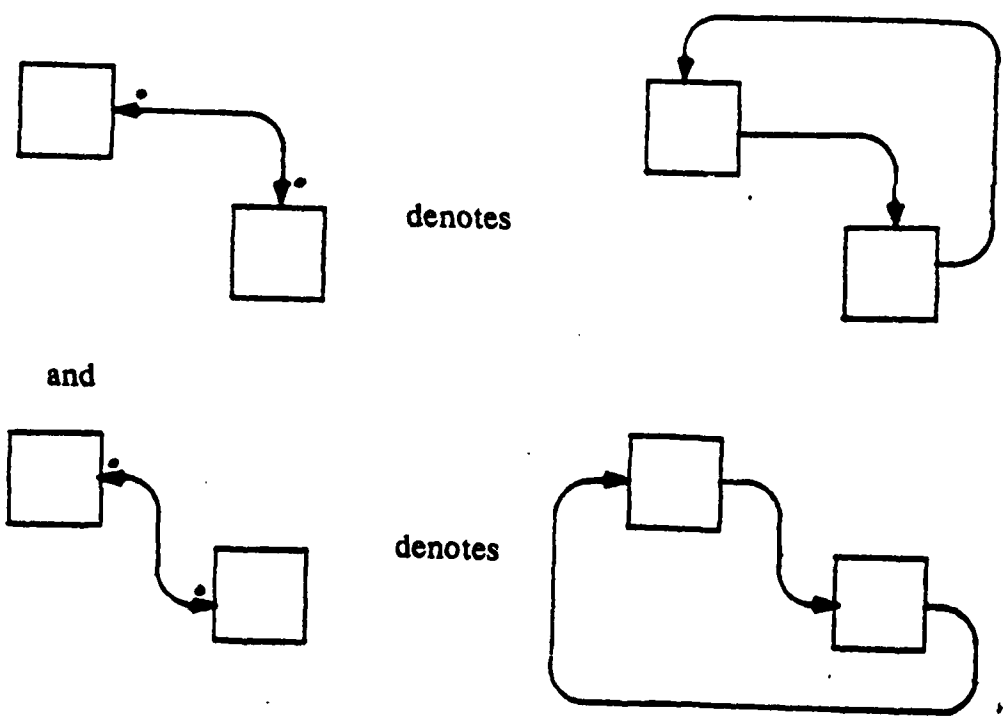
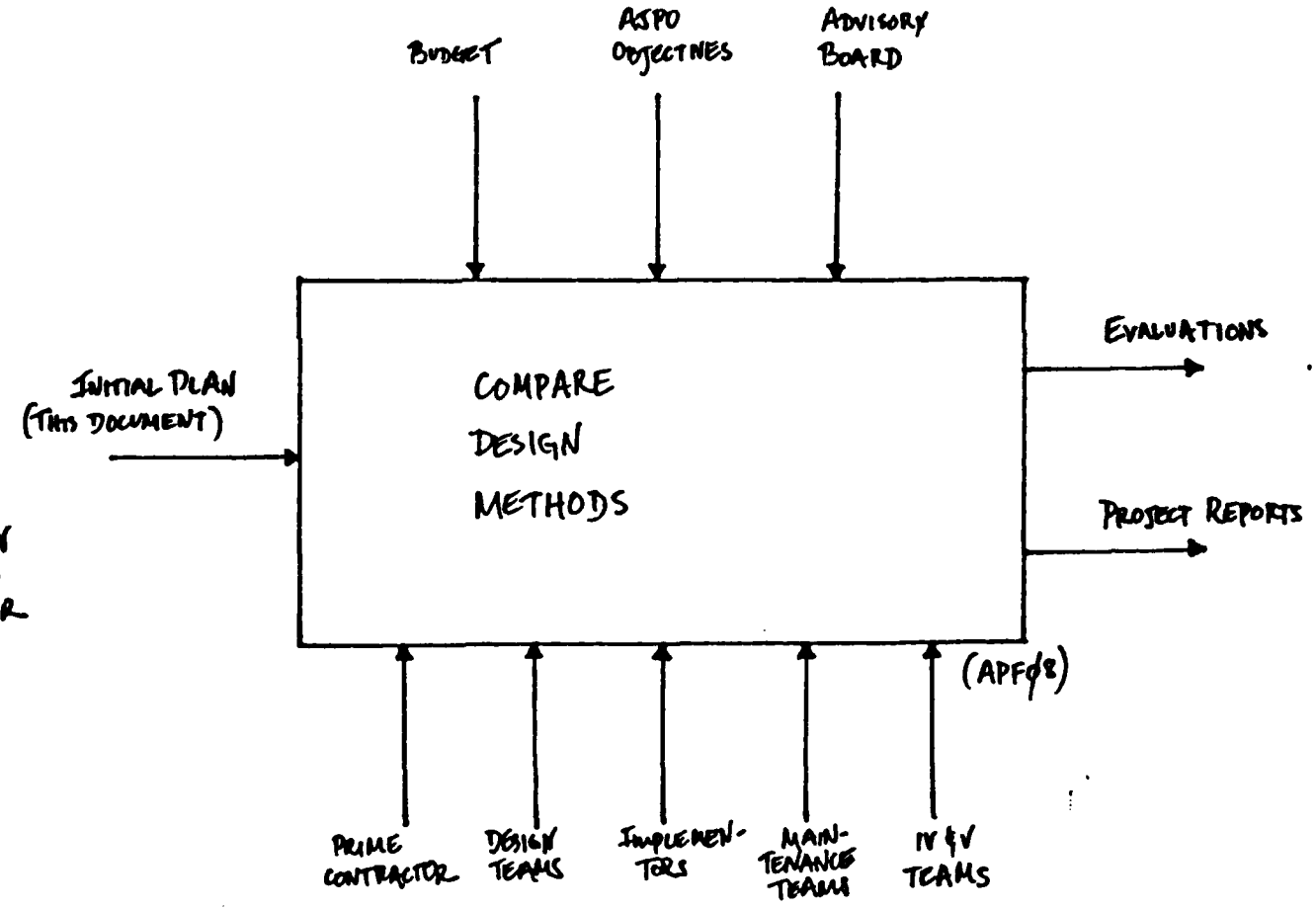


Figure 5.2.

SADT© DIAGRAM FORM ST098 9/75

USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 10/24/82 REV:	WORKING DRAFT RECOMMENDED PUBLICATION	READER DATE	CONTEXT:

1. GLOSSARY NOT YET PREPARED FOR THIS MODEL



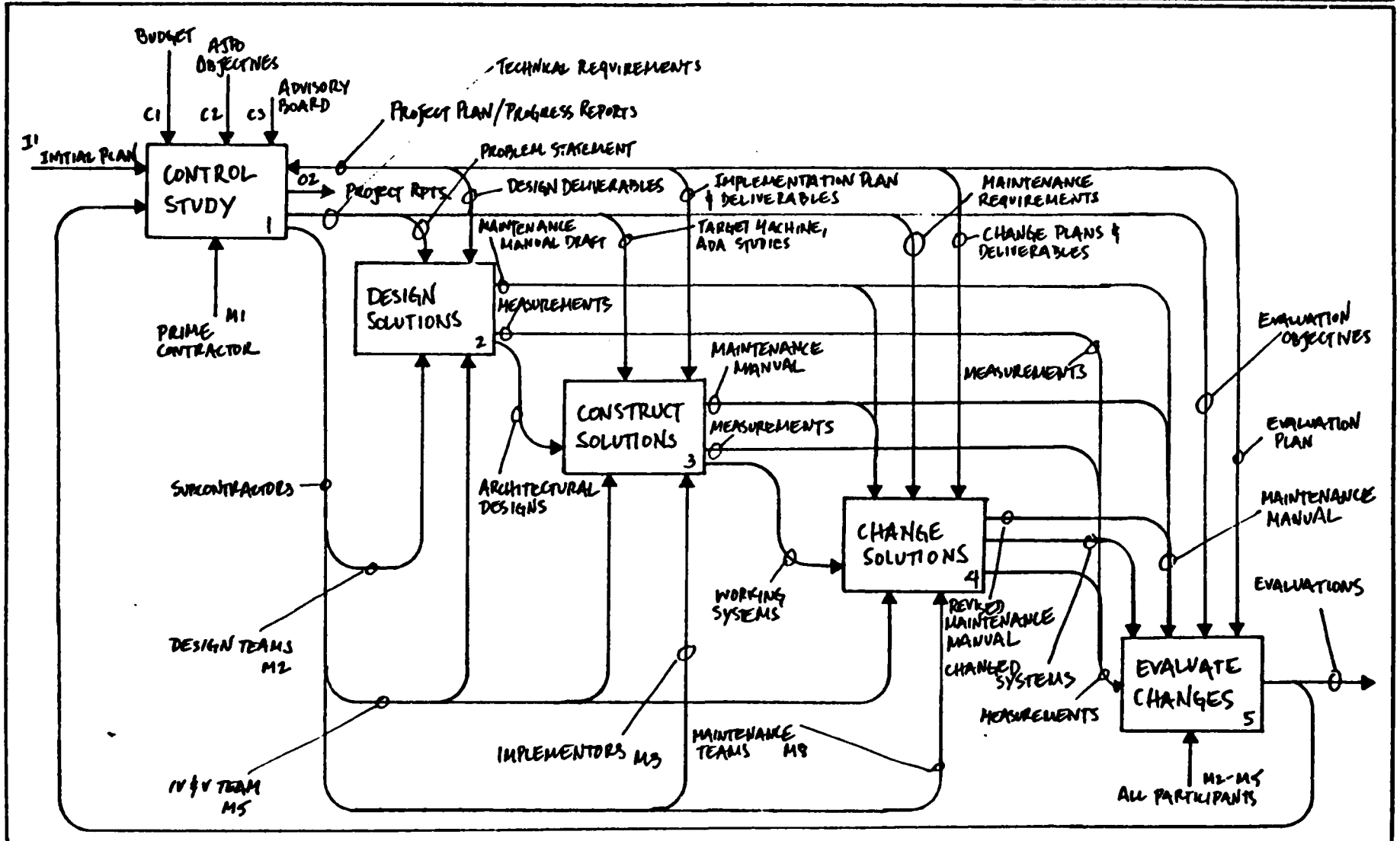
PURPOSE: PROVIDE MODEL OF DESIRED PROJECT ORGANIZATION TO PERMIT ACCURATE, DETAILED PLANNING BY PRIME CONTRACTOR

VIEWPOINT: PREPARERS OF INITIAL PLAN

NODE: A-0	TITLE: COMPARE DESIGN METHODS (CONTEXT)	NUMBER: APF07 (APF/8)

SADT® DIAGRAM FORM ST098 9/75

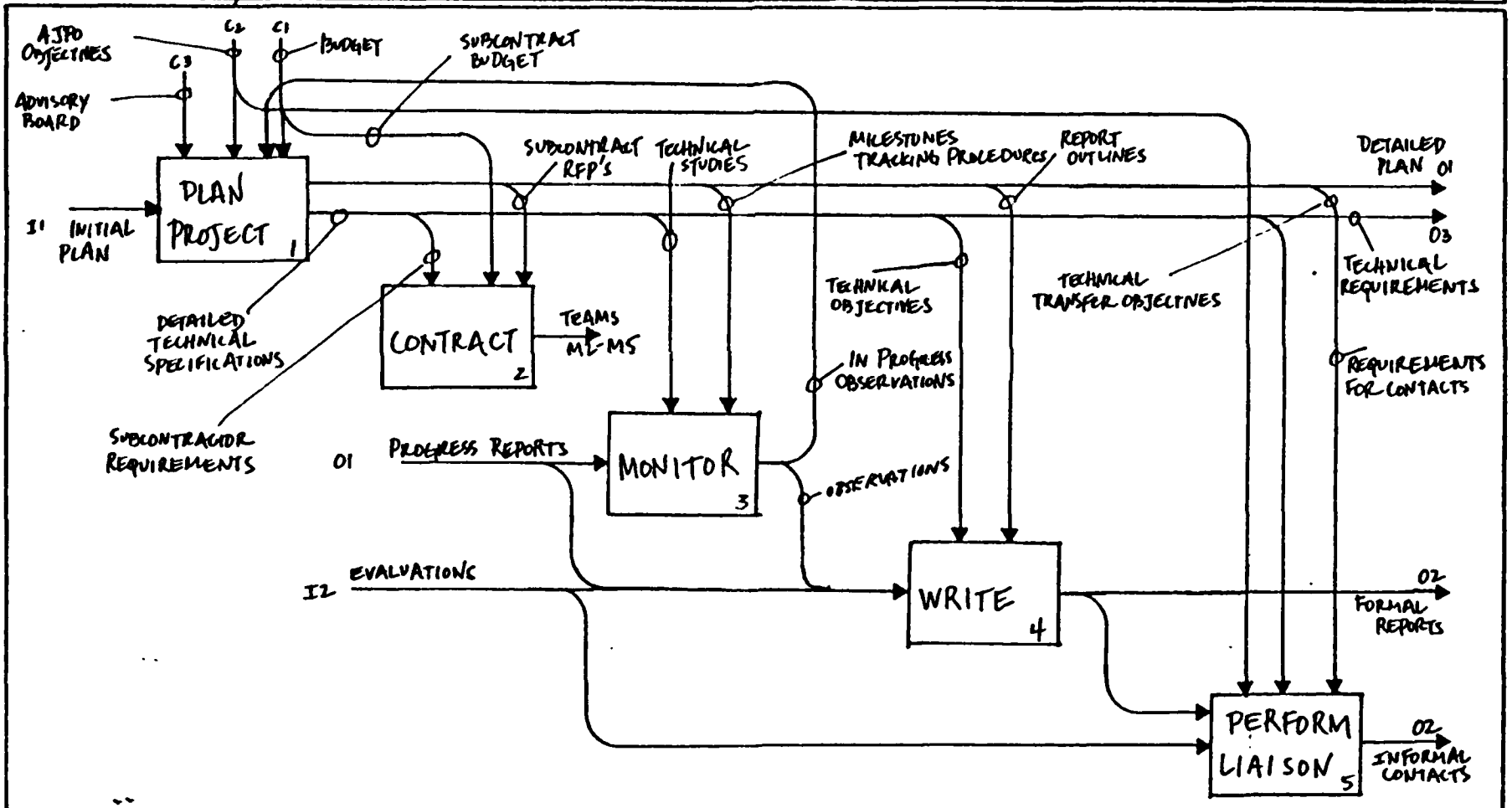
USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS	DATE: 10/24/82 REV:	WORKING / DRAFT / RECOMMENDED / PUBLICATION	READER DATE	CONTEXT:
	NOTES: 1 2 3 4 5 6 7 8 9 10				



NODE: A0	TITLE: COMPARE DESIGN METHODS	NUMBER: APF08 (APF04)

SADT® DIAGRAM FORM ST09 9/75

USED AT:	AUTHOR: PETER FREEMAN PROJECT: ADA METHODS NOTES: X 2 3 4 5 6 7 8 9 10	DATE: 10/24/82 REV:	WORKING DRAFT RECOMMENDED X PUBLICATION	READER DATE	CONTEXT:



NODE:	TITLE:	NUMBER:
A1	CONTROL STUDY	APF09 (APF06)